Conception d'un système de partage de données adapté à un environnement de Fog Computing

Bastien Confais

UNIVERSITE BRETAGNE LOIRE

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Bastien CONFAIS

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université de Nantes
sous le sceau de l'Université Bretagne Loire

École doctorale : Mathématiques et Sciences et Technologies de l'Information et de la Communication (MathSTIC)

Discipline : Informatique
Unité de recherche : Laboratoire des Sciences du Numérique de Nantes (LS2N)

Soutenue le 10 juillet 2018
ED 601

Conception d'un système de partage de données adapté à un environnement de Fog Computing

JURY

Présidente :	**M^me^ Sara BOUCHENAK**, Professeur des Universités, INSA, Lyon
Rapporteurs :	**M. Frédéric DESPREZ**, Directeur de Recherche Inria, Grenoble **M. Marcelo DIAS DE AMORIM**, Directeur de Recherche CNRS, LIP6, Paris
Examinateur :	**M^me^ Sara BOUCHENAK**, Professeur des Universités, INSA, Lyon
Directeur de thèse :	**M. Benoît PARREIN**, Maître de Conférences titulaire de l'HDR, Université de Nantes
Co-encadrant de thèse :	**M. Adrien LEBRE**, Chargé de Recherche titulaire de l'HDR, Institut Mines-Télécom Atlantique, Nantes

Remerciements

Cela fut une belle opportunité que de pouvoir continuer mes études après mon cursus ingénieur. Ne me sentant, à cette époque, pas encore prêt à passer du monde de l'école au monde du travail, j'ai immédiatement accepté de travailler sur ce sujet qu'est le « Fog Computing ». Après avoir entrevu quelques solutions de stockage distribué lors de mon cursus ingénieur, continuer la découverte de ce domaine dans un contexte que je ne connaissais pas, dont je n'avais pas idée des problématiques associées fut réellement intéressant. M'appuyer sur mes connaissances et faire des rapprochements entre le domaine du réseau et le domaine de l'informatique distribuée fut un vrai plus pour moi. Le travail demandé n'a pas toujours été facile, et je ne serais certainement pas allé jusque-là sans l'aide de nombreuses personnes.

Je remercie tout particulièrement M. Desprez et M. Dias de Amorim qui m'ont fait l'honneur de rapporter cette thèse. Je les remercie non seulement pour leur relecture attentive de mon manuscrit mais également pour la pertinence de leurs remarques.

Je tiens à remercier également Mme Bouchenak pour avoir présidé mon jury de thèse.

Je remercie aussi mes deux encadrants, M. Parrein pour son soutien au quotidien, pour ses nombreuses idées et ses propositions mais aussi pour sa flexibilité dans l'organisation de mes journées ainsi que M. Lebre pour ses exigences, ses remarques toujours pertinentes, et sans qui les protocoles et les résultats expérimentaux seraient bien moins satisfaisants. Je les remercie également pour leur aide à la rédaction, pour leurs nombreuses relectures et pour avoir présenté mes travaux dans plusieurs conférences, chose que je ne pouvais pas faire et que je n'aurais pas réussi à faire moi-même.

Je n'oublie pas non plus M. Lehn et M. Ibrahim pour avoir suivi mes travaux année après année, m'avoir encouragé et m'avoir proposé de nouvelles pistes de travail.

Cette thèse n'aurait certainement pas aboutie sans l'aide de Mme Nachouki, non seulement pour m'avoir accepté dans son bureau pendant un an, pour m'avoir soutenu dans les moments difficiles mais également pour m'avoir proposé d'encadrer un projet avec des étudiants, me permettant de prendre du recul sur ce travail de thèse.

Un merci également à Dimitri qui s'est intéressé à mon travail, qui m'a donné de nombreux conseils, et à Alexandre pour ses relectures, ses idées d'améliorations et surtout pour avoir continué à travailler sur ma dernière contribution. Cela fait toujours plaisir de voir que mon travail intéresse d'autres personnes.

Je ne dois également pas oublier de remercier Mme Galais pour m'avoir enseigné

l'anglais, sans quoi je n'aurai pu réaliser cette thèse.

Un merci aux secrétaires du département informatique, Leila et Corinne pour leur bonne humeur et pour leurs discussions sur la cuisine et les voyages, en guise de pause pendant les longues journées.

Un grand merci également à Adam, Gáibhin et Seán pour leur gentillesse, pour m'avoir permis de décompresser pendant les week-ends et m'avoir motivé pour terminer cette thèse.

Enfin, je dois remercier maman pour m'avoir emmené à l'école tous les matins pendant toutes ces années, mais aussi pour avoir bien voulu discuter des problèmes, des difficultés et des aménagements avec mes encadrants, des sujets de discussion que je n'ose pas aborder moi-même. Merci également à ma tante pour m'avoir soutenu et aidé à préparer la soutenance. Un merci également à Caroline pour son aide en anglais mais aussi pour m'avoir accompagné à l'école d'été RESCOM, au Croisic l'an passé.

Table des matières

Introduction

Aujourd'hui, de nombreux services reposent sur les infrastructures de l'informatique en nuage (Cloud Computing), fournissant une puissance de calcul et de stockage considérable et permettant de répondre à la plupart des besoins des utilisateurs. Pourtant, en s'appuyant sur un nombre restreint de centres de données, ces infrastructures ne pourront pas satisfaire les nouvelles contraintes de l'Internet des Objets (IdO). Ces capteurs n'ayant pas les ressources internes suffisantes pour traiter leurs données et prendre des décisions, nécessitent une infrastructure externe accessible avec un temps de réponse le plus réduit possible. Les infrastructures actuelles d'informatique en nuage ne peuvent pas satisfaire ces contraintes. C'est pourquoi, en 2012, Cisco a proposé l'approche de Fog Computing consistant à déployer un nombre important de centres de données, répartis sur plusieurs sites, placés au plus près des utilisateurs [Bonomi et al. 2012]. Ces centres de données, appelés sites de Fog Computing comportent un nombre réduit de serveurs (environ une dizaine) et répondent aux requêtes. De nombreuses applications peuvent bénéficier de cette infrastructure, par exemple pour le traitement de données issues de capteurs, dans un contexte de ville ou d'usine intelligente, pour déployer des équipements réseaux virtuels au plus près des utilisateurs, au sein des réseaux cellulaires ou encore pour permettre le développement de solutions de réalité virtuelle. L'objectif de ce travail est de proposer un système de stockage pour les infrastructures d'informatique en nuage bas ou Fog Computing.

Afin de permettre le développement de telles infrastructures, nous devons développer des mécanismes de gestion des ressources de calcul, de stockage et de réseau. C'est pourquoi nous cherchons à réaliser une solution de stockage dans lequel les utilisateurs soient capables d'accéder aux données depuis n'importe quel site de Fog Computing, avec le temps d'accès le plus faible possible, et ceci, sans avoir besoin de connaître l'endroit où elles sont stockées.

La première partie de cette thèse est consacrée à présenter un état de l'art. Dans le Chapitre 1, nous revenons sur l'historique de l'informatique utilitaire, à partir des grappes de calculs jusqu'à l'introduction de l'informatique en nuage. Nous détaillons les raisons du développement de chacune des approches ainsi que leurs avantages et inconvénients. Après cela, nous introduisons les nouvelles approches, visant à apporter de la puissance de calcul près des utilisateurs, à la périphérie du réseau. Nous présentons le concept de Fog Computing, consistant à déployer de petits centres de données, sur différents sites géographiquement répartis. Nous détaillons les caractéristiques et particularités d'une

telle approche.

Dans le Chapitre 2, nous nous concentrons sur les solutions de stockage distribuées : des approches reposant sur un serveur central aux approches totalement distribuées. Nous en présentons les principaux mécanismes ainsi que les concepts sur lesquels ces solutions de stockage reposent.

Enfin, dans le Chapitre 3, nous établissons une liste de cinq caractéristiques nécessaires pour qu'une solution de stockage soit adaptée à l'environnement de Fog Computing. Nous voulons *(i)* favoriser la localité des données, c'est-à-dire que les données stockées doivent être placées le plus proche de l'endroit où elles ont été générées. Nous voulons également *(ii)* confiner le trafic réseau au sein des sites de Fog Computing et limiter le trafic échangé entre les sites. La troisième caractéristique *(iii)* consiste à tolérer les pannes du réseau : lorsqu'un site se retrouve isolé des autres, les données qu'il stocke doivent rester accessibles aux utilisateurs de ce site. Enfin, *(iv)* la solution de stockage proposée doit supporter la mobilité des utilisateurs entre les sites et *(v)* doit passer à l'échelle, c'est-à-dire qu'elle doit être capable de fonctionner avec un grand nombre d'objets connectés, un grand nombre de sites et doit pouvoir stocker une quantité toujours plus importante de données. Nous analysons en quoi les solutions de stockage proposées jusqu'à présent ne sont pas adaptées aux infrastructures de Fog Computing.

La seconde partie de ce manuscrit détaille nos contributions. Notre première contribution, présentée dans le Chapitre 4, consiste en une analyse expérimentale des solutions de stockage existantes déployées dans un environnement de Fog Computing. Avant de développer notre solution, nous voulons vérifier si un logiciel existant peut être utilisé dans un tel environnement. Nous montrons qu'InterPlanetary FileSystem (IPFS), une solution de stockage reposant sur le protocole BitTorrent et une table de hachage distribuée (DHT) pour stocker la localisation des données, fournit les meilleures performances. Toutefois, plusieurs éléments nécessitent d'être améliorés : des trafics réseaux entre les sites sont générés pour chaque donnée accédée, augmentant la latence et empêchant le système de fonctionner lorsque le réseau est partitionné.

Le Chapitre 5 est consacré à la présentation théorique de nos améliorations. Notre seconde contribution vise à limiter le trafic réseau échangé entre les sites lors de la lecture d'un objet situé localement sur le site auquel l'utilisateur est connecté. Dans cette situation, la table de hachage distribuée est utilisée pour localiser l'objet, générant un trafic réseau entre les sites de Fog et impactant les temps d'accès. Nous proposons de coupler IPFS à un système de fichiers distribué qui est déployé de façon indépendante sur chaque site. De cette façon, les données stockées par un nœud sont mises en commun avec les données stockées par les autres nœuds du site. Les nœuds d'un site peuvent alors accéder à l'ensemble des données stockées sur celui-ci sans recourir à l'utilisation de la table de hachage distribuée pour les localiser.

Dans notre troisième contribution, également présentée dans le Chapitre 5, nous cherchons à limiter le trafic réseau lors de l'accès à des données stockées sur un site distant. En effet, un des principaux inconvénients de l'utilisation d'une table de hachage distribuée est que celle-ci ne fournit pas de localité : accéder à une donnée située sur un site distant demande de préalablement contacter un premier site stockant la localisation

de la donnée avant de contacter le site distant pour télécharger la donnée. Ce site stockant la localisation de la donnée est déterminée grâce à une fonction de hachage et peut donc correspondre à n'importe quel site du réseau. Cela peut mener à des scénarios où localiser la donnée nécessite plus de temps qu'accéder à la donnée elle-même. Pour palier cela, nous proposons de remplacer la table de hachage distribuée par une approche inspirée du protocole *Domain Name System* (DNS). Les sites sont organisés en un arbre respectant la topologie physique du réseau et chaque site interroge successivement ses ancêtres pour localiser un objet. Une fois l'objet localisé, des réplicas de métadonnées sont créés dynamiquement sur le chemin qui a mené à localiser l'objet en question. Interroger en premier les sites proches du site courant permet d'améliorer les temps de localisation mais cela permet également de confiner le trafic réseau dans une partie de celui-ci.

Enfin, le Chapitre 6 est consacré à l'évaluation expérimentale de nos propositions sur les plateformes Grid'5000 et FIT/IoT-Lab. Nous montrons que coupler IPFS à un système de fichiers distribué permet d'améliorer les temps d'accès aux objets stockés localement sur le site d'environ 34% en moyenne tout en réduisant le trafic réseau inter-sites échangé. En revanche, cela n'améliore en rien les accès lorsque la donnée accédée est stockée sur un autre site. Nous évaluons également notre approche de localisation des données, reposant sur un arbre des plus courts chemins et favorisant l'interrogation de serveurs situés à une faible latence. Nous montrons que l'utilisation d'un tel arbre permet de réduire les temps de localisation d'environ 20 %.

Dans une dernière section de ce Chapitre 6, nous cherchons à réaliser une expérimentation plus réaliste, impliquant de réels objets connectés, plutôt que des clients émulés. L'objectif est de vérifier comment l'ensemble des contraintes liées aux ressources limitées des objets connectés impactent les temps d'accès. Pour cela, nous essayons d'interconnecter les plateformes Grid'5000 et FIT/IoT-Lab. La plateforme Grid'5000 nous permet d'émuler les serveurs d'une infrastructure de Fog Computing tandis que la plateforme FIT/IoT-Lab fournit des capteurs mesurant des paramètres de l'environnement, comme la température.

Enfin, nous concluons ce manuscrit et donnons une liste de perspectives pour poursuivre ce travail.

Ces travaux ont donné lieu à plusieurs publications. Tout d'abord, l'analyse expérimentale a été présentée à la conférence IEEE Cloudcom en 2016 [Confais et al. 2016a] avant de faire l'objet d'une publication au journal Transactions on Large-Scale Data and Knowledge-Centered Systems (TLDKS) [Confais et al. 2017d] dans une version étendue. La première contribution sur IPFS a été présentée à la conférence IEEE ICFEC en 2017 [Confais et al. 2017b]. La gestion de la localisation s'appuyant sur un arbre est en cours de soumission à la conférence IEEE GlobeCom 2018 [Confais et al. 2018a]. Enfin, notre expérimentation utilisant conjointement les plateformes Grid'5000 et FIT/IoT-Lab a fait l'objet d'une présentation lors de l'école de printemps FIT/G5K [Confais et al. 2018e].

Contexte & État de l'art

Introduction de la première partie

L'informatique utilitaire a beaucoup évolué depuis plusieurs décennies, en partant des grappes de calculs qui ont vu le jour dans les années 1970 à l'informatique en nuage que nous connaissons aujourd'hui.

Dans le permier chapitre, nous proposons un tour d'horizon de ce passé, afin de comprendre les origines du Fog Computing, pourquoi ce paradigme a été proposé et quelles sont les infrastructures similaires. Après une rapide présentation des grappes et des grilles de calcul, nous nous concentrons sur les nouvelles approches telles que l'Edge Computing, le Mobile Edge Computing ou le Fog Computing. Nous nous concentrons ensuite sur les particularités de cette dernière architecture et nous présentons nos hypothèses de travail.

Dans un second chapitre, nous nous concentrons plus particulièrement sur le stockage. Nous listons et présentons les différentes architectures et solutions qui ont été proposées au fil du temps. Des solutions reposant sur un serveur central comme NFS, aux approches réparties adaptées aux grilles de calcul, tout en passant par les solutions développées pour du calcul à haute performance.

Dans le dernier chapitre nous nous concentrons sur le stockage dans un environnement de Fog Computing. Nous listons les caractéristiques que nous pensons être essentielles pour une solution de stockage fonctionnant dans un tel environnement. Nous regardons ensuite dans la littérature, si les solutions de stockage existantes satisfont ces caractéristiques.

1

Informatique utilitaire : quelles infrastructures après le Cloud ?

Sommaire

Les infrastructures de l'informatique en nuage sont un élément majeur de l'Internet d'aujourd'hui. Avec leur puissance de calcul et leur capacité de stockage quasi-infinie, elles permettent aux utilisateurs d'accéder à leurs données et d'exécuter des traitements depuis n'importe quel lieu de la planète. De nombreuses évolutions que nous détaillons dans les sections suivantes ont été nécessaires pour aboutir à la création des infrastructures d'informatique en nuage telles que nous les connaissons aujourd'hui.

1.1 Petit historique de l'informatique utilitaire

La puissance de calcul et l'espace mémoire fournie par un seul ordinateur n'ont jamais été suffisants par rapport à la quantité de calculs que nous avons à effectuer au quotidien. Il a été proposé de relier les ordinateurs entre eux, afin de créer de « gros » ordinateurs (« super ordinateurs »), beaucoup plus puissant et permettant de fournir des ressources à un plus grand nombre d'utilisateurs [Higbie 1973].

Cette informatique « distribuée » a beaucoup évolué ces dernières années. Le concept de grappe est apparu au début des années 1970. Une grappe était organisée de la façon suivante : une machine frontale permet à l'utilisateur de soumettre ses tâches, un ensemble de nœuds de stockage et de nœuds de calcul permettent de réaliser les tâches demandées. Les nœuds sont reliés entre eux par un réseau local. Une machine de sauvegarde permet de stocker le travail en cours lorsque la grappe traite les travaux d'un autre utilisateur. Ce type de grappe a plusieurs inconvénients, notamment pour augmenter sans-cesse leur capacité de calcul. Au-delà d'une certaine taille, cela pose des problèmes de place dans les bâtiments, de fourniture d'électricité ou encore de climatisation. Ensuite, les utilisateurs doivent partager le même environnement. En d'autres mots, il ne peut pas y avoir par exemple d'utilisateurs qui utilisent chacun leur propre système d'exploitation : les utilisateurs doivent travailler avec les programmes et les bibliothèques installés sur les machines de la grappe.

Aujourd'hui, les grappes sont encore utilisées pour du calcul à haute performance (HPC), « par exemple ». Le Top 500[1] est un classement listant les 500 grappes les plus puissantes du monde. Actuellement, la première du classement est le *Sunway TaihuLight* qui fournit une puissance de calcul de 93 Pflops (derrière Tianhe-2, avec une puissance de 33.9 Pflops)

Le concept de « grille » est apparu dans les années 1990. Une grille est un ensemble de grappes réparties géographiquement sur un territoire et interconnectées (soit par un réseau dédié ou soit par le réseau Internet). Ce réseau d'interconnexion a une latence beaucoup plus élevée que la latence du réseau interne de chaque grappe. Cette latence élevée apporte de nombreuses difficultés, comme par exemple, le fait de pouvoir partager des données entre les différents sites. Accéder à des données situées dans une autre grappe de la grille devient une opération coûteuse. Des systèmes de fichiers comme Xtreemfs ont été inventés pour ce cas d'usage précis [Hupfeld et al. 2008].

[1] http://top500.org/

Dans le même temps, l'organisation des grappes s'est modifiée : les grappes ont été rendues utilisables comme un seul ordinateur avec les « systèmes à image unique » [Morin et al. 2004]. Une surcouche de bas niveau permet de fusionner l'ensemble des nœuds. Par exemple, lorsque l'utilisateur invoque la commande `ps`, les appels systèmes sont modifiés de sorte à retourner la liste de tous les processus fonctionnant sur l'ensemble des nœuds de la grappe. De telles implémentations ont été proposées par OpenSSI[2] ou encore openMosix[3]. L'inconvénient de ce genre de système est la lourdeur de la maintenance. Chaque ajout dans le noyau du système d'exploitation entraîne une modification du logiciel.

Dans le même temps, la quantité de données à traiter a augmenté significativement et dans un contexte de BigData, transférer les données vers les serveurs de calcul n'était plus envisageable. Il a donc été envisagé de déplacer les traitements sur les serveurs stockant les données plutôt que l'inverse. De cette manière, les nœuds de calcul et de stockage ont été fusionnés. Dans ce domaine, des implémentations comme Hadoop [White 2009] ou des surcouches comme Spark [Zaharia et al. 2010] ou Flint [Sharma et al. 2016] sont utilisées.

Aujourd'hui, l'utilisation de machines virtuelles dans les infrastructures d'informatique en nuage permettent une grande flexibilité. Les machines virtuelles permettent à chaque utilisateur d'utiliser un environnement différent et peuvent également fonctionner sur différentes architectures. Cela simplifie également l'utilisation des ressources puisqu'il est possible de mettre en pause les machines virtuelles, de les déplacer d'une machine physique à une autre, etc. Des stratégies de placement des données et des machines virtuelles existent pour optimiser l'utilisation des ressources et adapter les ressources mises à dispositions aux besoins des utilisateurs [Blagodurov et al. 2015]. Toutefois, les fournisseurs d'informatique en nuage ou acteurs de Cloud Computing ont chacun leurs infrastructures qui sont indépendantes (une machine virtuelle située chez Amazon ne sera pas déplacée chez Google par exemple).

1.2 Les besoins de l'Internet des Objets (IdO)

De plus en plus d'objets de la vie quotidienne et de capteurs sont connectés à l'Internet. Cela permet à la fois de fournir de nouvelles fonctionnalités aux utilisateurs mais aussi d'optimiser l'utilisation de ressources, par exemple pour fluidifier le trafic routier à l'échelle d'une ville [Gubbi et al. 2013]. Cisco avait estimé qu'en 2020, plus de 50 milliards d'objets seraient connectés à l'Internet [Evans 2011]. Bien que nous savons aujourd'hui que cette estimation est très surévaluée [Nordrum 2016], le nombre d'objets connectés n'a pas encore cessé de croître.

Ces objets ont tous des caractéristiques très diverses. Certains sont mobiles (*i.e.*, les robots de services légers), certains sont contrôlés par un utilisateur humain quand d'autres n'interagissent qu'avec d'autres objets connectés (communications *Machine-to-Machine* ou M2M). Certains ne sont que des capteurs qui effectuent une mesure régulièrement

[2] http://openssi.org/
[3] http://openmosix.sourceforge.net/

quand d'autres ont une capacité d'action sur leur environnement, comme par exemple un véhicule qui s'arrête après avoir reçu une instruction [Zanella et al. 2014]. Cependant, beaucoup ont des besoins de calcul et de stockage à faible latence afin de pouvoir prendre des décisions rapidement. Par exemple, une voiture autonome et connectée a besoin que les données issues de ses capteurs soient traitées le plus rapidement possible afin que la voiture puisse réagir avant qu'un accident ne se produise [Bonomi et al. 2012]. Pourtant, pour des raisons de coût, ces objets ont bien souvent une capacité de calcul et de stockage limitée et n'ont pas les ressources nécessaires pour traiter les données qu'ils collectent. La solution communément utilisée est de recourir à une infrastructure externe comme les infrastructures d'informatique en nuage que nous détaillerons dans la prochaine section.

1.3 Une taxonomie des infrastructures

De nombreuses architectures sont proposées pour pallier le manque de ressources sur les objets connectés tout en permettant un passage à l'échelle et des calculs à faible latence.

1.3.1 Informatique en nuage ou Cloud Computing

L'« informatique en nuage » ou Cloud Computing, est composée d'un nombre très important de serveurs répartis dans un nombre restreint de centre de données. Les technologies de virtualisation permettent d'attribuer à la demande des utilisateurs ces ressources quasi-infinies [Zhang et al. 2015].

Aujourd'hui, à l'ère du *Big Data*, ces infrastructures sont utiles pour réaliser des calculs sur de grandes quantités de données, pour les agréger mais aussi pour les archiver [Agrawal et al. 2011]. Dans un environnement d'Internet des Objets, l'informatique en nuage permet de fournir des ressources de calculs et de stockage à ces derniers. Les objets connectés y envoient les données collectées et reçoivent en réponse le résultat du traitement demandé [Biswas et al. 2014]. Les fournisseurs d'informatique en Nuage peuvent fournir des services de différents niveaux d'abstractions comme le *Software as a Service* (SaaS) consistant à héberger et à maintenir l'ensemble d'une application pour un utilisateur, ou l'*Infrastructure as a Service* (IaaS) consistant à fournir uniquement des machines virtuelles que l'utilisateur pourra configurer lui-même.

Cependant, de par leur architecture centralisée, les infrastructures de « Cloud Computing » ne sont pas adaptées aux besoins de l'Internet des objets. Premièrement, la position des centres de données, éloignée des utilisateurs et des objets connectés ne lui permet pas de répondre avec un faible temps de latence. Cela rend son utilisation incompatible dans un certain nombre de cas d'utilisations pour lesquels ce critère est essentiel [Bonomi et al. 2012]. Certains auteurs vont jusqu'à dire que la latence pour joindre un serveur situé sur un site de « Cloud Computing » est élevée et imprévisible [Zhang et al. 2015] (supérieure à 100 ms). La seconde raison est qu'avec peu de centres de données et un nombre toujours croissant d'objets connectés, le nombre d'objets connectés à chaque centre est toujours plus important. Cela accroît la charge réseau, particulièrement au niveau des liens qui convergent vers ces infrastructures [Zhang et al. 2015] et rend difficile

le passage à l'échelle. Pour donner un ordre de grandeur, Shi *et al.* [Shi et al. 2016] ont indiqué qu'en 2019, les objets connectés produiront 500 zetta-octets de données alors que le réseau qui mène aux infrastructures de Cloud ne peut actuellement faire transiter que 10.4 Zo. Cela est illustré sur la Figure 1.1(a) où l'infrastructure d'informatique en nuage est la source ou la destination de tous les transferts de données.

1.3.2 Edge Computing

L'« Edge Computing » est l'une des solutions apportées à ces problèmes. L'idée générale est de réaliser les calculs, non plus sur l'infrastructure de « Cloud Computing » mais sur des serveurs situées en bordure du réseau, au plus près des utilisateurs et des objets connectés.

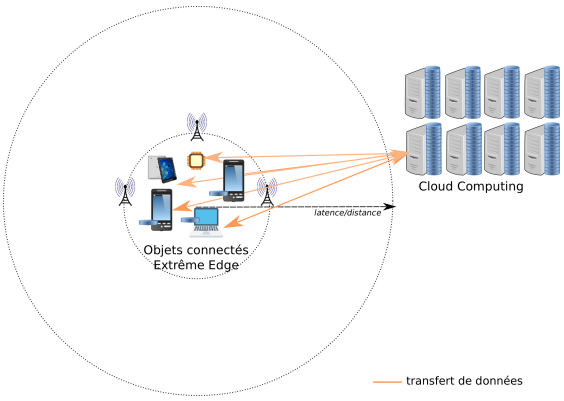
Pour cela, des serveurs sont déployés près des utilisateurs, en périphérie du réseau [Jararweh et al. 2014 ; Garcia Lopez et al. 2015 ; Shi et al. 2016]. Dans le cas d'un réseau cellulaire, les serveurs sont déployés près de chaque « station de base ». Les utilisateurs utilisent alors les ressources fournies par les serveurs situés dans la cellule à laquelle ils sont connectés. Ces serveurs effectuent les calculs qui nécessitent de faible temps de réponse et transmettent au Cloud ceux qui nécessitent des ressources plus conséquentes. Une telle architecture est également utile lors du stockage de données : l'utilisateur interroge un serveur situé à la périphérie du réseau. Si ce dernier n'a pas la ressource demandée, la requête est transmise à l'infrastructure Cloud. Lorsque les données sont retournées par le Cloud, celles-ci sont mises en cache afin de pouvoir répondre directement aux futures requêtes. Cela est illustré par la Figure 1.1(b). Les serveurs situés à la périphérie du réseau ont des ressources moins importantes que celles du Cloud mais suffisantes pour effectuer certains calculs ayant besoin d'une faible latence.

1.3.3 Extrême Edge Computing

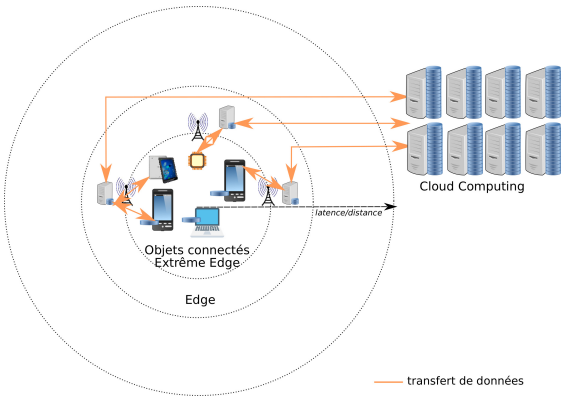
Le modèle d'« Extrême Edge Computing » a le même objectif que l'Edge Computing : traiter les requêtes au plus près des utilisateurs afin de limiter les sollicitations de l'infrastructure Cloud. Toutefois, la mise en œuvre est différente : au lieu de placer des nouveaux serveurs près des utilisateurs, ce sont les utilisateurs qui mettent en commun et partagent leurs ressources à l'aide de protocoles pairs-à-pairs (P2P) ou ad-hoc [Klemm et al. 2003]. Les calculs et le stockage sont donc effectués sur les périphériques (objets connectés) eux-mêmes. La Figure 1.1(c) montre une infrastructure d'Extrême Edge Computing. Les différences entre ces différents modèles d'infrastructure sont résumées par Yi *et al.* [Yi et al. 2015a].

1.3.4 Mobile Cloud Computing

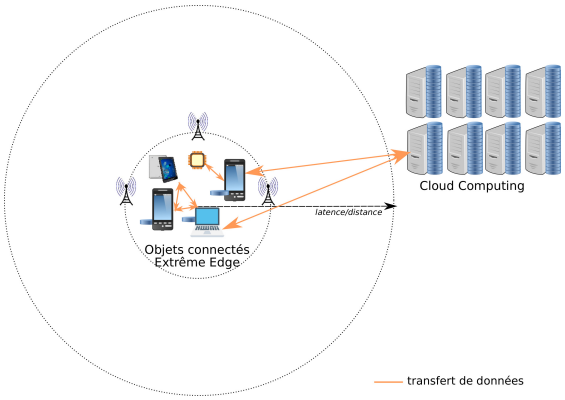
Ces infrastructures permettent à des clients mobiles, ayant de faibles ressources de pouvoir bénéficier des ressources des infrastructures Cloud. Il existe principalement deux architectures permettant cela [Fernando et al. 2013]. La première consiste à déployer des serveurs (appelés « Cloudlets ») près des utilisateurs situés au bord du



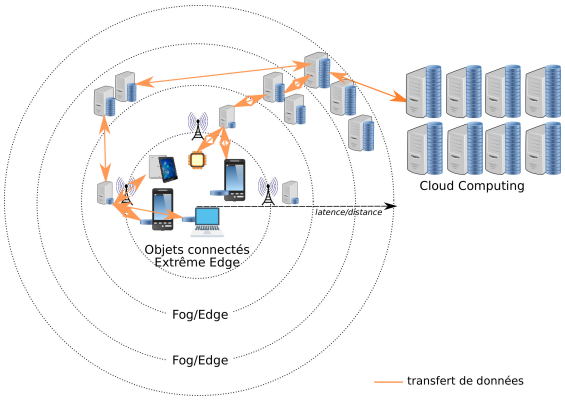
(a) – Approche Cloud Computing



(b) – Approche Edge Computing



(c) – Approche Extrême Edge Computing



(d) – Approche Fog Computing

FIGURE 1.1 – Exemple d'infrastructure de Cloud Computing (a), d'Edge Computing (b) et d'Extrême Edge Computing (c) et de Fog Computing (d) montrant les interactions et les capacités de stockage de chaque équipement.

réseau. Cette approche est équivalente au Edge Computing ou encore au Mobile Edge Computing [Ahmed et al. 2016 ; Beck et al. 2014]. La seconde architecture consiste à mettre en commun les ressources des différents périphériques situés à proximité, afin d'avoir une puissance de calcul suffisante pour les applications. Cette seconde approche est également connue sous le nom d'« Extrême Edge Computing ». Des modèles hybrides, mélangeant ces deux approches ont également été proposés [Giurgiu et al. 2009 ; Jararweh et al. 2016].

1.3.5 Fog Computing

Pour plusieurs auteurs, les infrastructures de type « Mobile Cloud Computing » ne peuvent faire face aux contraintes de mobilité des objets connectés [Yannuzzi et al. 2014 ; Malandrino et al. 2016]. Ainsi sur de petites cellules ou lorsque l'objet connecté a une très grande mobilité, le serveur sur lequel les calculs sont déportés changera en permanence. Ce changement sera la cause de latences élevées lors de la soumission d'un calcul.

Les infrastructures informatiques « en nuage bas », « en brouillard » ou encore appelées infrastructures de Fog Computing ont été proposées par Cisco en 2012 [Bonomi et al. 2012] et sont aujourd'hui supportées par de nombreux industriels[4]. L'architecture proposée consiste à déployer, non pas des serveurs isolés près des utilisateurs mais de petits centres de données répartis en différents « sites » situés à la périphérie du réseau. Ces centres de données comportent classiquement une dizaine de serveurs et fournissent des ressources de calcul et de stockage aux clients situés en bordure du réseau. De par leur taille plus importante par rapport à l'Edge Computing, les sites de Fog supportent une meilleure mobilité des utilisateurs tout en améliorant les temps de réponse par rapport à une infrastructure de type « Cloud ». Afin de simplifier le déploiement, il a été proposé de placer les serveurs dans les points de présence de l'Internet [Lebre et al. 2013]. Dans la Figure 1.1(d), chaque équipement accède à un serveur situé dans son environnement proche. Ce serveur effectue certains calculs localement et délèguent ceux qui sont trop coûteux à un autre serveur situé un peu plus loin de l'utilisateur final. De cette façon, les calculs sont répartis dans l'infrastructure. Les serveurs proches des utilisateurs s'occupent des traitements peu coûteux mais qui nécessitent un faible temps de réponse, jusqu'au Cloud Computing qui effectue les calculs très gourmands en ressources mais pour lesquels les temps de réponses sont moins importants.

Les infrastructures de Fog peuvent également être hiérarchiques, avec une architecture de « Cloud Computing » en haut de cette hiérarchie [Bonomi et al. 2014 ; Firdhous et al. 2014 ; Byers et al. 2015]. L'idée générale est que plus un site de « Fog » est « haut » dans la hiérarchie, plus les ressources qu'il met à disposition sont importantes. En contrepartie, la latence pour atteindre ce site est élevée. Le « Cloud » fournit une capacité de stockage et de calcul quasi illimitée au prix d'une latence élevée tandis que le site de Fog le plus proche de l'utilisateur fournit avec une très faible latence, une faible capacité de calcul et de stockage. L'intérêt d'une telle architecture est que lorsqu'un site de Fog a besoin de

[4] https://www.openfogconsortium.org/

données ou d'exécuter des calculs sur le « Cloud », l'infrastructure de « Cloud » n'est pas atteinte directement. De nombreux sites sont traversés, permettant de mettre en cache les données, de les agréger voire de réaliser certaines portions de calculs [Aazam et al. 2014]. De ce fait, le « Cloud » reçoit beaucoup moins de données que lorsqu'il est utilisé directement, permettant un meilleur passage à l'échelle. Dans certains cas, tous les calculs sont effectués avant que l'infrastructure de « Cloud » ne soit atteinte, réduisant le temps de calcul. La figure 1.1(d) illustre cette hiérarchie. L'idée générale du Fog est d'être capable d'effectuer des calculs de *Big Data* avec des contraintes de latence [Bittencourt et al. 2017].

Une telle architecture comporte d'autres avantages, notamment d'un point de vue de la vie privée et de la sécurité [Stojmenovic et al. 2014 ; Yi et al. 2015c]. Confier ses données à un serveur situé proche de soi permet de limiter le nombre d'attaquants potentiels que nous pourrions rencontrer sur le chemin. De même, chaque serveur ne gérant les données que d'un petit nombre d'utilisateurs, cela permet de limiter l'impact en cas de compromission de l'un d'entre eux. Enfin, d'autres auteurs évaluent le gain énergétique potentiel en redirigeant les requêtes vers le site de Fog ou le centre de données ayant l'empreinte carbone la plus faible au moment considéré [Jalali et al. 2016 ; Gao et al. 2012].

1.3.6 Hypothèses de travail

Dans ce manuscrit, nous considérons principalement une hiérarchie avec un seul niveau de Fog : les clients situés en bordure du réseau, une couche de Fog et le Cloud en haut de la hiérarchie. Cette hiérarchie est illustrée par la Figure 1.2 Les clients, interrogent toujours le site de Fog le plus proche, atteignable, bien souvent à l'aide d'un lien radio, ayant une latence réseau (L_{Fog}) de l'ordre de 10 ms [Sui et al. 2016 ; Huang et al. 2012]. Nous considérons également que la latence réseau entre les différents sites de Fog est comprise entre 50 et 100 ms. Cela correspond à la latence moyenne d'un lien de type *Wide Area Network* [Markopoulou et al. 2006]. Enfin, joindre l'infrastructure de Cloud Computing nécessite 200 ms [Couto et al. 2014 ; Firdhous et al. 2014].

1.4 Exemples d'applications du Fog Computing

De nombreux cas d'utilisations qui nécessitent l'utilisation d'une architecture de type Fog Computing ont été proposés.

1.4.1 Des feux de signalisation intelligents à la ville intelligente

Plusieurs travaux [Bonomi et al. 2012 ; Bonomi et al. 2014 ; Stojmenovic et al. 2014] ont proposé d'utiliser une infrastructure de Fog Computing pour réguler le trafic routier. L'idée proposée est qu'un feu de signalisation envoie son état aux voitures proches afin que ces dernières ralentissent. L'état des feux et la position des véhicules est régulièrement envoyée vers une plateforme de Fog qui établit une politique globale afin que les usagers attendent le moins longtemps possible et aient la durée de trajet la plus courte. De façon

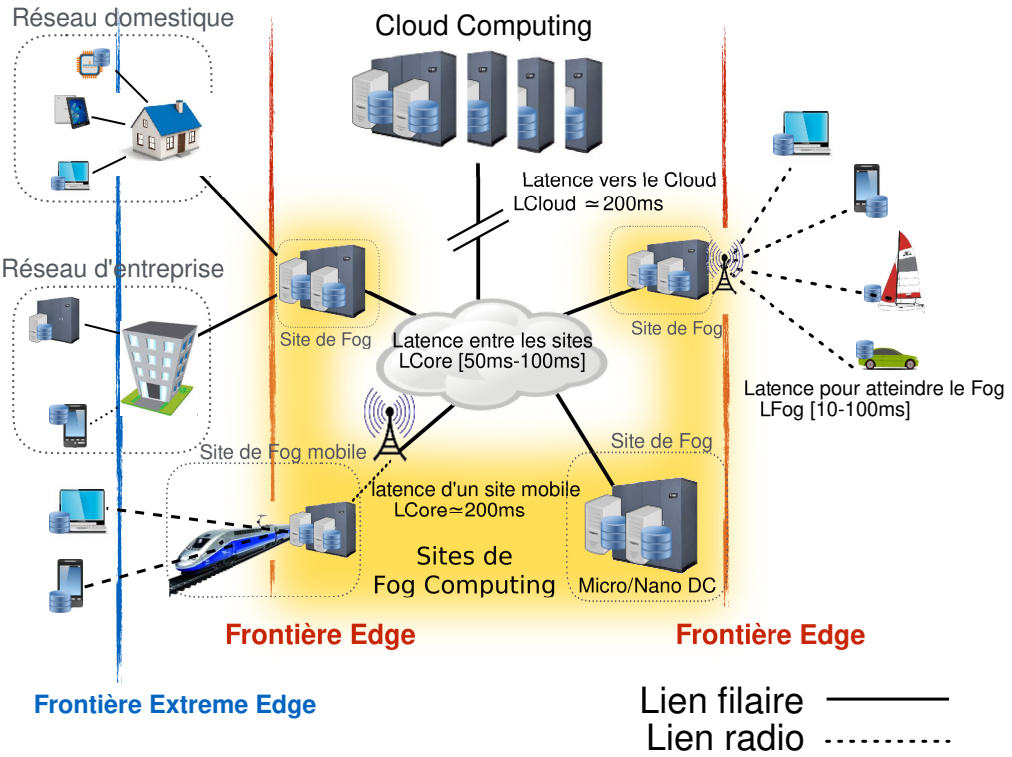


FIGURE 1.2 – Organisation générale des infrastructures de Cloud et de Fog Computing.

plus générale, le Fog a été proposé pour aider à la prise de décisions dans un réseau de véhicules (VANET) [TRUONG et al. 2015].

L'utilisation Fog a été proposée dans un contexte de « villes intelligentes » [TANG et al. 2015]. L'idée générale est d'utiliser les sites de Fog comme une plateforme de traitement pour les données collectées par des capteurs au sein de la ville.

1.4.2 Domaine de la santé

Dans le domaine de la santé, Dubey *et al.* [DUBEY et al. 2015] proposent d'utiliser le Fog dans la surveillance médicale. L'idée est que des capteurs mesurent différents paramètres de la personne. Transmettre cette grande quantité de mesures sur une plateforme de Cloud Computing pour l'analyser est très coûteux. L'idée proposée est d'introduire un Fog entre l'utilisateur et le Cloud afin d'agréger les données. Seul un agrégat des données rejoint le Cloud. L'infrastructure de Fog est également utilisée pour détecter des motifs dans les données qui signifient que la personne est en danger. L'objectif est de pouvoir être alerté le plus rapidement possible sans attendre que le calcul soit traité sur une plateforme de Cloud. D'autres auteurs [CAO et al. 2015 ; DASTJERDI et al. 2016b] proposent quant à eux d'utiliser le Fog pour détecter les accidents vasculaires cérébraux. Masip-Bruin *et al.* proposent quant à eux d'utiliser le Fog Computing dans le cadre de patients ayant des

difficultés à respirer. Des capteurs mesurent des données sur la personne, les transmettent à un site de Fog. À partir des données, le site de Fog prend la décision d'augmenter ou de réduire la quantité d'oxygène fournie à l'utilisateur [Masip-Bruin et al. 2016].

1.4.3 Réalité augmentée

Les infrastructures de type Fog Computing ne profitent pas seulement aux capteurs mais également aux périphériques utilisés par les utilisateurs. Des utilisations du Fog Computing ont aussi été proposées pour de la réalité augmentée [Satyanarayanan 2013 ; Yi et al. 2015b ; Dastjerdi et al. 2016a]. L'idée est d'utiliser le Fog pour analyser rapidement une image issue de l'environnement. Dans ce cas d'utilisation, seule la contrainte de latence rend l'utilisation du Cloud non appropriée. Plus généralement, le Fog Computing peut être utilisé pour fournir de la puissance de calcul à des terminaux qui en ont besoin. Par exemple, un téléphone portable peut avoir besoin de plus de mémoire qu'il n'en possède. Une idée est alors de déporter la mémoire utilisée par les applications inutilisées sur un serveur de Fog Computing [Yi et al. 2015a]. L'utilisation d'un serveur proche permet de rapatrier rapidement les pages lorsque celles-ci seront demandées par le système.

1.4.4 Traitement vidéo

Pour Satyanarayanan *et al.* [Satyanarayanan et al. 2015], le Fog pourrait être utilisé pour stocker des vidéos enregistrées par des caméras situées en bordure du réseau. Les flux vidéos sont conséquents et le Cloud ne pourrait pas supporter un grand nombre de caméras. Toutefois, leur proposition ne s'arrête pas là, la hiérarchie de Fog permettrait de réaliser des traitements sur les vidéos, et notamment créer une base de centralisée dans le Cloud qui permette de rechercher dans les vidéos stockées dans le Fog.

Un autre cas d'utilisation des infrastructures de Fog Computing est la reconnaissance de visage [Hu et al. 2017]. Les passants sont filmés par une caméra et les images sont transmises à un site de Fog afin de reconnaître en temps réel les personnes.

1.4.5 Mise en cache de contenu

La mise en cache de contenus est un cas d'usage très fréquemment proposé. L'idée générale est d'utiliser l'infrastructure de Fog pour réduire le temps d'accès à une donnée (comme un site web [Zhu et al. 2013]), en répliquant le contenu proche des utilisateurs mais également pour réduire la charge des serveurs stockant lesdites données et permettre un meilleur passage à l'échelle [Zeydan et al. 2016].

Pour Luan *et al.* [Luan et al. 2015], un site de Fog peut être placé dans un bus pour mettre en cache du contenu vidéo diffusé pendant que ce dernier roule. Le Fog peut être également utilisé dans un magasin pour stocker des informations utiles de façon très localisée.

Par exemple, lorsque des spectateurs enregistrent une vidéo dans un stade et veulent la partager entre-eux. En utilisant une infrastructure de type Cloud, cela non seulement

génère des temps de réponse élevés, génère une charge importante sur le Cloud (plusieurs centaines d'utilisateurs se connectent en même temps), mais aussi, cela occupe de façon importante les liens réseaux entre le stade et le centre de données hébergeant le service. Utiliser une infrastructure de Fog Computing permet de réduire les latences et de limiter la quantité de trafic réseau qui sort du stade.

1.4.6 Fonctions de virtualisation réseau

Un autre usage souvent proposé est d'utiliser une infrastructure de type Fog Computing pour déployer des fonctions réseau au plus près des utilisateurs [Vaquero et al. 2014 ; Yi et al. 2015b ; Vilalta et al. 2016 ; Chiang et al. 2016] L'approche consiste à virtualiser les équipements qui composent les réseaux informatiques (pare-feux, serveurs mandataires, routeurs, commutateurs) au sein des sites de Fog. Il n'est donc plus question d'aller déployer un pare-feu physique à tel ou tel endroit du réseau mais d'installer une machine virtuelle ayant ce rôle dans le site de Fog correspondant. Cette approche permet de créer des réseaux flexibles, reconfigurables et adaptés aux besoins des utilisateurs tout en garantissant une latence pour l'accès aux différents services. Placer des machines virtuelles accessibles avec une faible latence pour décharger des équipements mobiles a également été envisagé [Bittencourt et al. 2015].

1.4.7 Robotique et industrie du futur

L'Internet des Objets est aussi présent dans les usines et les processus industriels [Xu et al. 2014]. Dans cet environnement, le Fog Computing permet également de fournir de la puissance de calcul à des robots mobiles [Dey et al. 2016] afin de permettre une prise de décision rapide. Dans les usines du futur, le Fog peut être utilisé pour analyser les données issues de capteurs mais également pour contrôler la production de façon centralisée. Des machines virtuelles sont alors automatiquement déployées pour en contrôler le bon fonctionnement mais aussi pour les commander [Brito et al. 2016]. De même, pour Wan *et al.* [Wan et al. 2016], une infrastructure de Fog peut être utilisée pour produire des marchandises personnalisées selon les besoins de l'utilisateur. Par exemple, l'utilisateur donne les caractéristiques du produit qu'il souhaite sur le site web de l'entreprise puis le système de production est reparamatré afin que les machines fabriquent ce qui a été demandé.

1.5 Conclusions

Le Fog Computing a été introduit pour répondre à un besoin de calcul à faible latence que ne peuvent pas fournir les infrastructures de Cloud Computing mais également pour une question de passage à l'échelle : le Cloud ne pouvant faire face à l'explosion du nombre d'utilisateurs. Il est également intéressant de noter que les approches de Fog Computing ou d'Edge Computing sont très semblables et ne se démarquent que sur des aspects mineurs. Dit autrement, le Fog considère des ressources organisées de façon continue entre le Cloud et les utilisateurs tandis que l'Edge Computing ne considère bien souvent qu'un

niveau intermédiaire. En pratique, la plupart des solutions logicielles déployées pour de l'Edge Computing fonctionnent dans une infrastructure Fog et inversement. Aujourd'hui, les infrastructures de Fog devaient être introduites dans le réseau 5G afin de fournir une puissance de calculs aux téléphones portables [Yi et al. 2015a ; Yi et al. 2015b][Kitanov et al. 2016].

Dans le chapitre suivant, nous allons nous intéresser aux contraintes que nous rencontrons lorsque nous voulons stocker des données dans une infrastructure de Fog Computing. Nous nous interrogeons également sur l'endroit où stocker les données, c'est-à-dire, sur les stratégies de réplication. Également, gérer la localisation des données, c'est-à-dire, être capable de retrouver l'emplacement d'une donnée, n'est pas un problème trivial dans une architecture de Fog Computing. Nous présentons les solutions existantes en montrant leurs lacunes.

2

Les solutions de partage de données dans une infrastructure distribuée

Sommaire

Dans ce chapitre, nous allons nous intéresser au fonctionnement de différentes solutions de stockage qui ont été proposées pour les différentes infrastructures de l'informatique utilitaire que nous avons présentées dans le Chapitre 1.

2.1 Systèmes de stockage pour les grappes

Les données sont souvent stockées dans des fichiers, au sein de systèmes de fichiers en réseau qui mettent à disposition de l'ensemble des utilisateurs, une arborescence de dossiers et de fichiers [Levy et al. 1990]. Tous les utilisateurs accèdent simultanément à la même arborescence et le système de stockage s'occupe de gérer les permissions, les accès concurrents et la consistance. D'un point de vue de l'utilisateur, une fois le système de stockage monté dans son arborescence virtuelle (*Virtual FileSystem* ou VFS), les accès se font comme des accès aux fichiers stockés localement. Les accès aux fichiers se font avec certaine granularité (4 Ko, 8 Ko).

Historiquement, ces systèmes de fichiers en réseau étaient fournis par un seul et unique serveur de stockage centralisé auquel tous les nœuds de calcul se connectaient. Des protocoles réseau comme NFS [Beame et al. 2003] (Network FileSystem) ou CIFS [Hertel 2003] (Common Internet FileSystem) étaient les plus couramment utilisés. Toutefois, le principal problème de cette approche est sa capacité à passer à l'échelle : le serveur central n'a pas le débit réseau nécessaire et devient un goulet d'étranglement lorsque beaucoup de clients souhaitent accéder à des données simultanément. De même, augmenter l'espace de stockage fourni n'était pas quelque chose de facile puisqu'il fallait bien souvent changer le serveur par un serveur permettant d'insérer plus de périphériques de stockage ou bien l'arrêter pour remplacer les disques durs par des disques de plus grande capacité.

Afin de pallier ces inconvénients, une nouvelle architecture fut proposée. Plusieurs serveurs de stockage sont utilisés simultanément et un serveur de métadonnées est utilisé en guise d'annuaire, pour stocker la localisation de chaque fichier. Pour accéder à une donnée, le client interroge le serveur de métadonnées pour déterminer la localisation de celle-ci, puis interroge le serveur de données stockant l'information souhaitée. Un tel protocole est décrit par le diagramme de la Figure 2.1

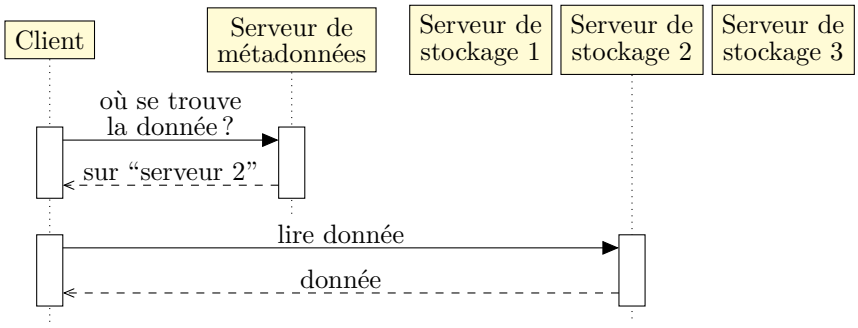


Figure 2.1 – Diagramme de séquence montrant comment un client accède à une donnée dans un système de stockage pour le calcul à hautes performances.

L'intérêt d'une telle approche est non seulement de répartir la charge entre plusieurs serveurs de stockage mais également de permettre une meilleure parallélisation des accès. Bien que le serveur de métadonnées reçoit autant de requêtes que dans le cas d'un serveur centralisé, ce dernier a beaucoup moins de données à stocker et à transmettre aux clients,

lui permettant d'avoir une charge plus faible qu'un serveur de stockage centralisé. Cette approche a été utilisée dans de nombreux systèmes de fichiers comme PVFS [Carns et al. 2000], Lustre [Donovan et al. 2003], ou encore RozoFS [Pertin et al. 2014].

Une telle approche propose un passage à l'échelle horizontal (*Scale-Out NAS*), cela signifie que pour augmenter de l'espace de stockage, il suffit simplement de rajouter des serveurs de stockage. Cela est très différent des approches proposant un passage à l'échelle vertical qui consistent à augmenter la capacité de stockage de chaque nœud en y ajoutant des disques durs, voire en remplaçant les machines par des machines ayant plus de ressources. L'avantage principal d'une telle approche est que l'ajout de serveurs de stockage n'augmente pas seulement la capacité de stockage du système de fichiers mais augmente également la capacité totale du réseau, ce qui permet à un nombre toujours plus important de clients d'accéder à l'infrastructure.

Cependant, l'ajout de serveurs augmente également la probabilité de défaillance du système. Afin de fournir une disponibilité des données conséquente, les pannes doivent être tolérées tout en permettant aux utilisateurs de continuer à accéder aux données. Elles ne sont plus l'exception mais la norme. En effet, les durées de vie des périphériques de stockage (Mean Time To Failure) ne sont pas infinies et plus il y a de périphériques de stockage, plus la probabilité que l'un d'entre eux soit défaillant est élevée. Certains systèmes comme Lustre [Donovan et al. 2003] proposent de répliquer les données (généralement 3 fois) tandis que d'autres comme RozoFS [Pertin et al. 2014] proposent d'utiliser des codes à effacement.

Plusieurs stratégies de réplication existent. Parmi les plus simples et les plus utilisées, une première consiste à faire que ce soit le client qui écrive l'ensemble des réplicas lorsqu'il modifie un objet tandis qu'une second stratégie consiste à ce que ce soit les serveurs de stockage qui réalisent la création et la modification des réplicas, permettant au client de ne réaliser l'écriture que d'un seul d'entre eux [Weil et al. 2006a].

Ces solutions s'appuient sur un réseau haute performance, fournissant des débits très élevés et de faibles latences. Elles ne sont par conséquent, pas adaptées pour fonctionner dans un environnement géographiquement distribué, où les sites sont interconnectés par un réseau de type WAN (*Wide Area Network*). Dans ce contexte, le site stockant le serveur de métadonnées concentrera une grande partie du trafic réseau, pouvant occasionner la saturation des liens mais aussi une indisponibilité des données dans le cas où ce serveur ne peut pas être atteint.

2.2 Systèmes de stockage pour les grilles

Comme nous l'avons détaillé dans la Section 1.1, la particularité des grilles est qu'elles sont réparties sur différents sites, interconnectés par un réseau de type WAN, avec un faible débit et une forte latence. Les systèmes de fichiers distribués présentés précédemment ne sont pas adaptés à un tel environnement car accéder à un serveur de stockage situé sur un site distant demande beaucoup de temps, limitant les performances fournies aux utilisateurs.

Des systèmes de fichiers spécifiques ont été développés comme XtreemFS [Hupfeld

et al. 2008], GFarm [Tatebe et al. 2010], GlobalFS [Pacheco et al. 2016] ou certaines adaptations de Lustre [Simms et al. 2007 ; Walgenbach et al. 2010]. Afin de permettre un fonctionnement dans un environnement multi-sites ces systèmes de fichiers répliquent les données sur les sites où les utilisateurs y accèdent. En introduisant cette localité des données, cela permet aux utilisateurs de bénéficier d'accès rapides aux données, une fois que ces dernières ont été relocalisés.

Toutefois, cette approche ne résout pas le problème des métadonnées. Solliciter un serveur centralisé pour stocker la localisation des fichiers est coûteux et empêche le déploiement des systèmes de fichiers sur plusieurs sites. En effet, contacter, à l'aide de liens à forte latence, un premier site distant pour localiser le fichier puis un second site distant pour accéder à la donnée demande beaucoup de temps ce qui limite très fortement les performances du système. En plus de cette difficulté d'accès au serveur de métadonnées qui peut être atteignable avec une latence plus élevée que les données elles-mêmes, ces systèmes ont également des problèmes de consistance entre les sites.

Fondamentalement, il n'y a pas de grandes différences architecturales entre ces solutions et les solutions développées pour les grilles. L'architecture est la même. Les principales adaptations sont que les solutions de stockage pour les grilles s'appuient généralement sur des protocoles qui traversent facilement les pare-feu réseau comme le protocole HTTP mais également, comme nous venons de le dire, ces solutions utilisent une stratégie de placement et de réplication des données qui favorise la localité.

2.3 Systèmes de stockage pour les infrastructures de Cloud Computing

Traverser un système de fichiers est coûteux puisque par exemple, à chaque niveau, les permissions de l'utilisateur doivent être vérifiées. De plus, l'utilisation d'un serveur de métadonnées est l'élément le plus problématique, à la fois car cela crée un point unique de défaillance dans le réseau mais aussi parce qu'il peut également devenir le facteur limitant dans les performances du système.

Afin de permettre un meilleur passage à l'échelle, les solutions de stockage pour le Cloud reposent sur un stockage en mode objet. L'idée du stockage par objets est de proposer un espace de nommage « à plat ». La notion de dossiers disparaît, permettant de limiter les accès aux métadonnées, et chaque objet est uniquement identifié par son nom [Nightingale et al. 2012]. Les systèmes fournissent une interface basique et simple aux utilisateurs : les utilisateurs ne peuvent que demander à connaître le contenu d'un objet ou écrire un objet. Cette interface d'accès aux données est proposée notamment par Amazon S3 qui s'appuie notamment sur la solution de stockage Dynamo [DeCandia et al. 2007]. Cassandra est quant à lui une solution de stockage utilisée dans l'infrastructure de Facebook pour indexer les messages échangés entre les utilisateurs [Lakshman et al. 2010] . Enfin, Rados [Weil et al. 2007] est également très souvent utilisé pour stocker les images de machines virtuelles au sein de la solution de virtualisation Openstack [Fitfield 2013]. Cette interface simple d'accès aux données permet de proposer des solutions de stockage qui ne reposent pas sur un serveur de métadonnées centralisées, supprimant le goulet

d'étranglement que cela peut engendrer. Ces solutions peuvent donc fonctionner dans des environnements de plus large échelle tout en fournissant de meilleures performances tout que les solutions de stockage développés pour les grilles de calcul. Nous détaillerons par la suite, pourquoi ces systèmes sont des candidats potentiels pour les infrastructures de Fog Computing.

Nous notons toutefois que quelques systèmes de fichiers ont été proposés pour fonctionner dans des infrastructures de Cloud Computing. C'est le cas par exemple de Glusterfs [Davies et al. 2013] qui ne repose pas sur un serveur de métadonnées mais sur des fonctions de hachage. L'inconvénient majeur de cette solution est que les clients doivent connaître l'ensemble de la topologie, sans qu'aucun mécanisme ne soit proposé pour automatiser cela. D'autres personnes ont proposé de développer des systèmes de fichiers au-dessus des approches par objets, afin de retrouver une interface utilisable par les applications. C'est le cas notamment de CephFS [Weil et al. 2006a] qui s'appuie sur Rados ou encore Cassandra FileSystem (CFS) qui s'appuie sur la solution Cassandra [Luciani 2012]. Du fait de leur implémentation considérant chaque bloc du fichier comme un objet, ces solutions sollicitent fortement le système de stockage par objets sous-jacent.

2.4 Réseaux de distribution de contenus

La problématique du stockage de données n'a pas été seulement étudiée sous une approche « système », mais elle a également été étudiée sous un aspect réseau. Dans les réseaux, la problématique est de stocker des données accédées par un grand nombre d'utilisateurs simultanément, tout en évitant une saturation des liens connectant la solution de stockage à ces derniers.

Les réseaux de diffusion de contenus (Content Distrbution Network ou CDN) consistent à mettre en cache des données issues d'une source sur des serveurs situés au plus près des utilisateurs. Cela a deux principaux intérêts : *(i)* accéder à un serveur proche permet de réduire les temps d'accès aux ressources demandées *(ii)* cela permet de réduire la charge du serveur central puisque ce dernier n'a plus qu'à envoyer le contenu une fois au plus, à chaque serveur de cache. Cette technique permet d'améliorer le passage à l'échelle. Akamai [Nygren et al. 2010] ou Cloudflare font partie des réseaux de distribution de contenus publics les plus connus mais nombreux sont ceux déployés en interne par les entreprises comme Google [Gill et al. 2007], afin d'améliorer leur qualité de service et d'usage.

Il n'y a pas d'architecture typique pour ces réseaux. Différentes architectures ont été proposées et sont aujourd'hui utilisées. Bien que les serveurs soient situés physiquement près des utilisateurs, l'architecture du réseau dont nous parlons est un réseau de recouvrement.

2.4.1 Réseau de recouvrement organisé en arbre

La première approche consiste à organiser le réseau en arbre. La source de données se trouve à la racine de l'arbre. Les clients interrogent un serveur de cache. Lorsque celui-ci

ne contient pas la donnée recherchée, il contacte à son tour son serveur parent. La requête remontre jusqu'au parent qui contient un réplica de la donnée, ou dans le pire des cas, jusqu'à la source de données située à la racine de l'arbre. Chaque nœud qui demande l'objet à son parent mettra en cache cette dernière lorsque le parent lui répondra. De cette façon, les données les plus fréquemment demandées sont plus à même de se trouver dans les serveurs caches situés près des clients

Ce mécanisme simple permet d'accéder au réplica de données le plus proche.

Le problème de cette approche est que la charge n'est pas équilibrée entre les nœuds, un nœud proche des feuilles de l'arbre sera bien plus souvent sollicité qu'un nœud proche de la racine. Pour faire face à cela, Karger *et al.* [Karger et al. 1997] proposent d'utiliser un arbre différent pour chaque objet, afin que toutes les requêtes ne suivent pas le même chemin. Pour cela, une fonction de hachage est utilisée. Dans l'approche proposée, l'empreinte de l'identifiant de la donnée à accéder est utilisée pour déterminer l'arbre. À partir de cette empreinte chaque nœud détermine son nœud parent. Le problème de cette approche est qu'elle ne prend pas en compte la latence réseau. Le nœud parent d'un serveur peut physiquement être atteignable avec une latence élevée. De la même façon, les clients sont alors affectés à un serveur choisi aléatoirement.

Au lieu de construire un arbre en utilisant une fonction de hachage, d'autres approches [Tran et al. 2005 ; Ganguly et al. 2005] proposent de prendre en compte le débit réseau et de construire un arbre dans lequel le débit est suffisant pour garantir que la donnée pourra être transférée en un temps relativement faible. Les algorithmes proposés reviennent à chercher un arbre couvrant de poids maximum.

Tout comme dans les approches de stockage distribuées, certaines approches considèrent le placement des réplicas dans un réseau CDN [Kangasharju et al. 2002]. Au lieu de répliquer sur tous les nœuds du chemin sur lequel la donnée est demandée, des approches proposent de prendre en compte d'autres contraintes comme la capacité de stockage des nœuds ou la disponibilité de ces derniers [Khan et al. 2009].

2.4.2 Solutions reposant sur une table de hachage distribuée

Une autre architecture possible pour construire un réseau de diffusion de contenus est de reprendre les architectures utilisées dans les soluions de stockage distribués. Bien que certaines peuvent localiser les objets avec une approche centralisée, beaucoup d'autres reposent sur un mécanisme de placement distribué, comme nous pouvons trouver dans les solutions de stockage par objets. Ainsi CoralCDN [Freedman et al. 2004] ou encore UsenetDHT [Sit et al. 2005], deux réseaux de diffusion de contenu, reposent tous les deux sur une table de hachage distribuée pour stocker les données. Les nœuds ont également un cache local des derniers objets demandés afin de servir les clients plus rapidement. Les inconvénients d'une telle approche sont les mêmes que ceux des systèmes de stockage distribué, à savoir que le nœud devant stocker un réplica de données est déterminé à partir d'une fonction de hachage, limitant la localité.

Comme dans les systèmes de stockage distribués, certaines approches stockent localement un réplica de la donnée et ne stockent que la localisation des données dans une structure de données telles qu'une table de hachage distribuée. C'est le cas par exemple

de FlowerCDN [El Dick et al. 2009] qui utilise une table de hachage distribuée entre les sites et un serveur de métadonnées centralisé au sein de chaque site.

La principale différence de ces systèmes par rapport à un « véritable » système de stockage distribué est que les réplicas sont immuables. Les données en cache ne sont accédées qu'en lecture et ne sont jamais modifiées, ce qui simplifie les stratégies de réplication et la gestion de la consistance des données. Cela permet également de créer des réplicas à la volée. Dans les systèmes de fichiers distribué, le niveau de réplication est fixe, c'est-à-dire que les données sont répliquée deux ou trois fois par exemple. En revanche, dans les approche de type CDN, le nombre de réplicas de chaque donnée dépend de sa popularité.

2.4.3 Protocoles pairs à pairs

Au lieu d'organiser le réseau en arbre, des protocoles pairs à pairs comme BitTorrent [Legout et al. 2005] peuvent être utilisés entre les nœuds. De cette façon, les données ne sont pas téléchargées d'un seul et unique nœud parent mais de plusieurs nœuds simultanément. Cela permet de réduire le temps de transfert ainsi que la charge des liens réseau tout en augmentant la disponibilité de la donnée.

Le protocole BitTorrent fonctionne de la façon suivante. Les données sont découpées en pièces de taille fixe (par exemple 64 Ko). Le nœud voulant télécharger des données, récupère la liste des identifiants des pièces composant la donnée qu'il souhaite télécharger. Un serveur central appelé *tracker* ou une table de hachage distribuée est interrogée pour connaître la liste des nœuds stockant les pièces demandées. Le client se connecte alors à plusieurs nœuds stockant au moins une pièce et leur transmet la liste des pièces qu'il aimerait recevoir. Les pièces sont envoyées par blocs (par exemple de 512 octets) d'un nœud à l'autre. Au fur et à mesure que le client reçoit les morceaux de pièces dont il a besoin, il transmet des mises à jour de cette liste afin d'éviter de recevoir plusieurs fois les données. En effet, si deux nœuds stockent une même pièce, il est probable que les deux nœuds l'envoient au client puisque ce dernier demande à chaque nœud auquel il est connecté, la liste de **toutes** les pièces dont il a besoin. Il existe des mécanismes permettant de télécharger en premier les pièces rares, c'est-à-dire les pièces ayant le moins de réplicas et qui sont susceptibles de ne plus être disponibles en cas de panne d'un nœud ou de partitionnement du réseau. Nous noterons que les données dans BitTorrent, identifiées par leur empreinte, sont également immuables et ne peuvent pas être modifiées. Une telle approche est mise en œuvre dans des outils de synchronisation comme BitSync [Farina et al. 2014] ou Syncthing [Borg 2015].

Les protocoles pairs à pairs peuvent être utilisés exclusivement en bordure du réseau, entre les périphériques des utilisateurs [Palazzi et al. 2009] ou bien s'appuyer sur une infrastructure de Fog pour former un réseau de distribution de contenus hybride [Ghareeb et al. 2013 ; Xu et al. 2006] dans lesquels les clients n'interrogent pas directement le serveur de l'infrastructure de Fog mais essaient dans un premier temps de télécharger la donnée voulue depuis l'un de leurs voisins. Les clients ont également un rôle de serveur de cache, ce qui permet de réduire la charge des serveurs du réseau de diffusion de contenus. Xu *et al.* [Xu et al. 2006] proposent d'utiliser les serveurs du réseau de contenus pour y

placer le *tracker* nécessaire à l'échange de données entre les clients.

Afin de privilégier l'accès aux réplicas stockés sur des nœuds « proches » de l'utilisateur, plusieurs protocoles existent. Le protocole Vivaldi [Dabek et al. 2004b] permet de créer une « carte du réseau ». Une fois cette carte construite, il est alors possible de privilégier les nœuds situés à une faible distance. L'inconvénient de cette approche est que construire et maintenir la carte, génère une quantité importante de trafic sur le réseau.

Le protocole P4P [Xie et al. 2008 ; Kiesel et al. 2014] est quant à lui constitué d'un serveur central qui est interrogé par les nœuds voulant télécharger des données. Son rôle est de retourner une distance entre deux nœuds spécifiés. La distance peut être calculée de plusieurs façons et prendre en compte de nombreux paramètres tels que la latence réseau, l'homogénéité des débits sur le chemin, la sortie d'un système autonome (AS), etc. Les clients vont ensuite essayer de privilégier de télécharger les données depuis les nœuds ayant une faible distance. Cette approche intéresse fortement les opérateurs afin de confiner le trafic pair à pair au sein de leur réseau, évitant ainsi de toujours déployer des liens d'interconnexion avec les autres opérateurs de plus grandes capacités.

2.4.4 Réseaux centrés sur l'information (ICN)

Les réseaux centrés sur l'information (ou Information-Centric Networks ou ICN) et sa variante de Networking Named Content [Jacobson et al. 2009] est une technologie qui consiste à utiliser le réseau pour router les requêtes directement vers la machine stockant la donnée demandée. L'étape consistant à localiser la donnée avant d'y accéder a été supprimée. Le client envoie une requête dans le réseau à destination de la donnée qu'il souhaite accéder et non à destination d'une machine. Le réseau s'occupe alors de diriger celle-ci vers le serveur stockant la donnée qui répondra au client. Il est possible de considérer que l'ICN consiste, au lieu d'utiliser un serveur de métadonnées, à répartir celles-ci dans tout le réseau. Plusieurs mises en œuvre ont été proposées [Koponen et al. 2007 ; Jacobson et al. 2009 ; Hahm et al. 2017].

Une telle approche a été proposée pour fonctionner au sein d'un centre de données [Ko et al. 2012], au niveau d'une grille de calcul mais également dans un contexte distribué sur plusieurs sites et plus particulièrement dans un réseau de diffusion de contenu [Passarella 2012 ; Wires et al. 2017]. L'intérêt est de ne pas avoir à émettre de requête pour localiser la donnée et par conséquent de gagner du temps. Toutefois, la principale difficulté dans cette approche est que les routeurs du réseau doivent connaître l'emplacement de chaque réplica de données ce qui est coûteux en stockage.

Dans un réseau traditionnel, router le trafic internet demande beaucoup de ressources. Les routeurs d'aujourd'hui ont beaucoup de mal à fonctionner avec plus de 512 000 routes IPv4 [Shen 2014]. Pour limiter le nombre de routes, des méthodes d'agrégations consistant à utiliser la même route pour plusieurs destinations sont utilisées [Li 2011]. Dans IPv6, il est même proposé de ne pas annoncer de routes plus spécifiques qu'un préfixe de taille 32 ou 48 bits dans les tables de routage globales [Popoviciu et al. 2008] afin de limiter la taille de ces dernières. Les approches comme le protocole LISP (*Locator/Identifier Separation Protocol*) visent à séparer le rôle d'identificateur de l'adresse IP de son rôle permettant de localiser une machine. Dans ce contexte, de nombreuses propositions visant

à résoudre le problème de passage à l'échelle des tables de routage ont été proposées mais aucune ne semble satisfaisante [Jakab et al. 2010 ; Mathy et al. 2008].

Il est évident qu'en enregistrant dans chaque routeur, une route par objet, l'ICN n'est pas une approche qui pourra passer à l'échelle. Afin de réduire la quantité de routes à déployer, Fayazbakhs *et al.* [Fayazbakhsh et al. 2013] proposent de regrouper les données stockées aux mêmes endroits et d'annoncer seulement les routes correspondant aux différents groupes. L'inconvénient d'une telle approche est que l'utilisateur a besoin de connaître le groupe auquel la donnée fait partie pour y accéder, ce qui peut revenir à devoir connaître l'endroit où elle est stockée. Cela pose également des difficultés lors de la création d'un nouveau réplica puisque lorsque un réplica d'une donnée est ajoutée, cette donnée n'a plus la même localisation que les autres du groupe, obligeant à la changer de groupe.

2.4.5 Réseaux logiciels (SDN)

Les réseaux logiciels ou *Software Defined Networks* (SDN) sont généralement utilisés pour déployer une configuration réseau de manière centralisée [Qin et al. 2014]. L'utilisation d'une telle infrastructure a également été proposée pour maintenir les tables de routage au sein d'un ICN [Veltri et al. 2012]. Le Software Defined Network consiste à déployer le paramétrage d'un réseau de façon logicielle, notamment les routes des routeurs. Trois composants sont utilisés :

une base de données qui contient la description du service à fournir ;

des routeurs qui reçoivent leur table de routage par le réseau ;

un contrôleur centralisé qui déploie les routes de chaque routeur.

Dans le SDN, le routage ne se fait pas forcément uniquement sur le critère de l'adresse de destination d'un paquet mais peut se faire sur d'autres critères comme la source ou encore le type de paquet. Dans le cas d'un réseau centré sur l'information, la localisation des objets est stockée dans la base de données. Le contrôleur calcule ensuite la table de routage de chaque router et la déploie. Cela est illustré par la Figure 2.2.

Dans certaines approches, le contrôleur est également utilisé pour réaliser le placement des données (ou de machines virtuelles [Costa-Requena et al. 2015]). Dans ce cas, il est facile d'assimiler le rôle du contrôleur à celui d'un serveur de métadonnées, comme dans les solutions de stockage distribuées traditionnelles.

Nous noterons que le SDN a les mêmes difficultés de passage à l'échelle que celles mentionnées précédemment [Casado et al. 2012 ; Yeganeh et al. 2013] Ainsi, de plus en plus d'approches cherchent à utiliser plusieurs contrôleurs simultanément [Sun et al. 2013 ; Jiménez et al. 2014 ; Wan et al. 2016], ou encore, cherchent à placer ces derniers dans une infrastructure de Cloud Computing, là où les ressources virtuellement infinies [Kahvazadeh et al. 2017]. Une chose est sûre, le placement du ou des contrôleurs est un problème à part entière [Heller et al. 2012].

Aujourd'hui les réseaux centrés sur l'information sont encore un concept récent et de nombreux problèmes restent à résoudre bien que ceux-ci aient été identifiés [Kutscher

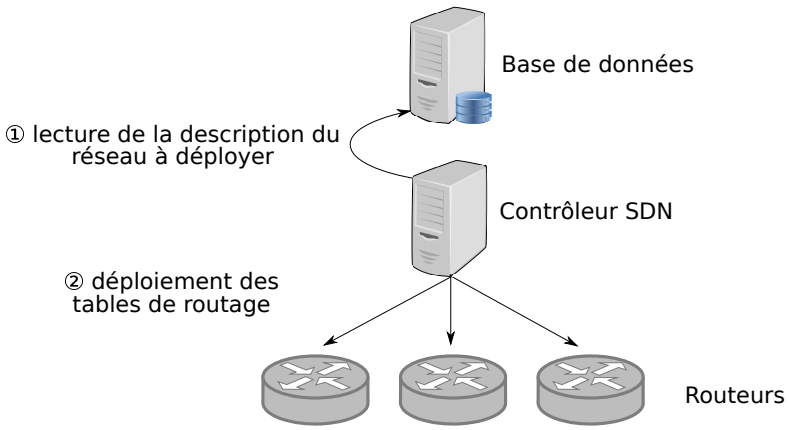


FIGURE 2.2 – Exemple d'architecture d'un système de SDN.

et al. 2016]. C'est pourquoi, ces approches ne semblent pas être largement déployées et qu'à notre connaissance il n'en n'existe pas encore de déploiement industriel d'un réseau de ce type.

2.5 Conclusions

Dans ce chapitre, nous avons présenté les solutions qui ont été proposées pour le stockage de données et le routage des requêtes dans différents environnements. De nombreuses stratégies ont été mises en œuvre pour réduire les temps d'accès, tolérer les pannes et localiser les données le plus rapidement possible. Toutefois, de nombreuses approches reposent sur un serveur central pour localiser les données, ce qui pose des problèmes de tolérance aux pannes et de performances, notamment si nous voulons déployer de telles solutions dans un environnement multi-sites. Seules les approches pour le Cloud et les approches de type *Content Delivery Network* permettent d'accéder à des données sans nécessiter de serveur central d'aucune sorte.

Finalement, tous ces systèmes peuvent être comparés selon trois critères : le passage à l'échelle, les performances et leur facilité d'utilisation par les applications. Une telle comparaison est illustrée dans la Figure 2.3.

La figure montre que les systèmes de fichiers sont plus facile à utiliser, du fait de leur compatibilité avec l'existant. Aujourd'hui, toutes les applications savent nativement enregistrer leurs données dans des fichiers. Les approches non distribuées comme les partages de type NFS ou CIFS s'intègrent et s'utilisent très facilement mais ne fournissent ni passage à l'échelle, ni performance. Dans les systèmes de fichiers distribués reposant sur un serveur de métadonnées centralisé , nous trouvons Lustre [DONOVAN et al. 2003] ou RozoFS [PERTIN et al. 2014] qui fournissent de très hautes performances, tout en supportant un nombre très important de serveurs de stockage mais qui ne peuvent pas être déployés dans un environnement multi-sites. Au contraire, XtreemFS [HUPFELD et al. 2008] qui a été spécifiquement conçu pour les grilles de calcul, fournit un support

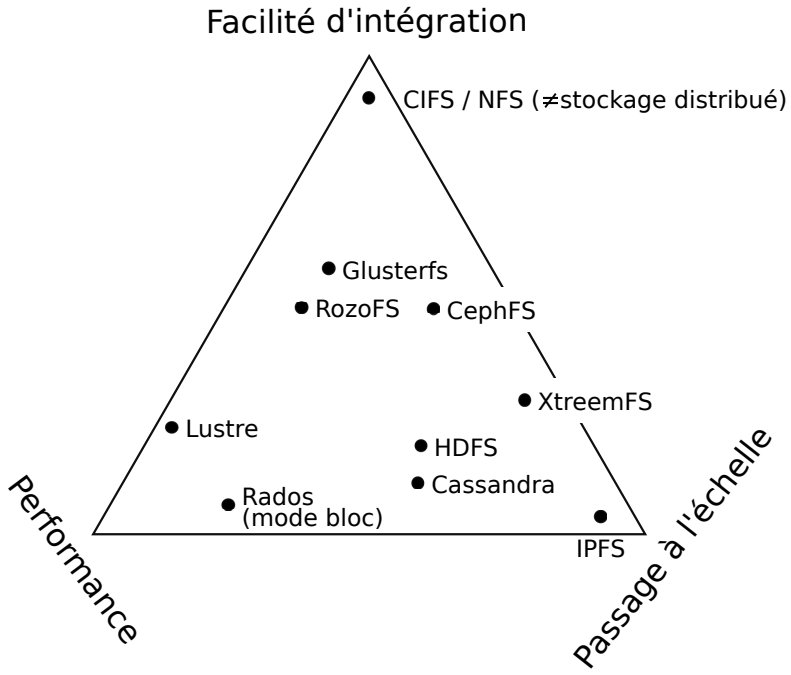


FIGURE 2.3 – Comparaison de différentes solutions de stockage distribuées, à savoir Glusterfs [DAVIES et al. 2013], RozoFS [PERTIN et al. 2014], CephFS [SEVILLA et al. 2015], Lustre [DONOVAN et al. 2003], XtreemFS [HUPFELD et al. 2008], HDFS [SHVACHKO et al. 2010a], Rados [WEIL et al. 2007] et Cassandra [LAKSHMAN et al. 2010]. CIFS [HERTEL 2003] et NFS [BEAME et al. 2003] sont également indiqués à titre indicatif.

multi-site au prix de faibles performances. Les solutions telles que CephFS [WEIL et al. 2006a] ou Glusterfs [DAVIES et al. 2013] proposent de remplacer le serveur de métadonnées par des fonctions de hachage. Enfin, les solutions par objets telles que Rados [WEIL et al. 2007], Cassandra [LAKSHMAN et al. 2010] ou encore IPFS [BENET 2014] sont plus difficiles à intégrer, mais permettent un meilleur passage à l'échelle avec une bien plus faible dégradation de performances que les systèmes de fichiers. Nous n'avons pas représenté sur la figure les approches reposant sur un réseau de distribution de contenus (CDN) car les caractéristiques en matière de performances et de facilité d'utilisation et d'intégration de ces approches dépendent énormément des implémentations sous jacentes. Il ne semble pas exister, à notre connaissance, de comparaison des performances des différentes approches qui peuvent être utilisées dans ce contexte. De plus, le fait est que les déploiements industriels d'architecture de type CDN ne sont pas souvent décrites publiquement où sont seulement décrites de façon assez générale sans rentrer dans les détails des mécanismes internes. Il est ainsi assez difficile de savoir comment fonctionnent précisément les réseaux de distribution de contenu déployés par des industriels comme Akamai [NYGREN et al. 2010], Cloudflare ou Google.

Il apparaît clairement qu'aucun système distribué ne fournit à la fois des performances,

un passage à l'échelle et une facilité d'utilisation. Nous avons donc besoin de clairement définir ce que nous attendons d'un système de stockage adapté pour le Fog, quels caractéristiques sont réellement nécessaires, afin de pouvoir choisir le système le mieux adapté.

3

Les spécificités du stockage de données dans une infrastructure de Fog Computing

Sommaire

Dans ce chapitre, nous nous intéressons aux particularités et aux caractéristiques que nous souhaitons avoir pour qu'une solution soit adaptée à un fonctionnement dans

un environnement de Fog Computing. Dans une seconde partie, nous vérifierons de manière théorique si les solutions de stockage proposées pour les infrastructures de Cloud Computing satisfont ces caractéristiques.

3.1 Caractéristiques attendues d'un système de stockage pour le Fog

Notre objectif est de créer un système de stockage unifié dans le Fog, cela signifie que chaque utilisateur doit pouvoir accéder à l'ensemble des données stockées, quel que soit le site auquel ce dernier est connecté. Nous avons défini plusieurs caractéristiques qu'un système de stockage idéal doit satisfaire.

3.1.1 Localité des données

L'idée de la localité des données est que les temps d'accès doivent être les plus faibles possibles. Pour cela, les données doivent toujours se trouver à l'endroit où elles sont utilisées. Lorsqu'un utilisateur écrit une donnée, il doit écrire sur le site de Fog le plus proche, c'est-à-dire, celui atteignable avec la plus faible latence possible. Il en est de même lors de la lecture : les données accédées doivent se trouver sur le site auquel l'utilisateur est connecté, toujours dans ce même but de minimisation des temps d'accès. Nous voulons donc limiter les accès pour lesquels la donnée accédée n'est pas présente sur le site.

3.1.2 Disponibilité des données

Nous voulons un système dans lequel les nœuds des sites puissent tomber en panne sans affecter les performances du système en général. Les pannes isolées doivent être dans la mesure du possible, gérées au sein du site dans lequel elles se sont produites, sans impacter les autres sites.

3.1.3 Confinement du trafic réseau

Le confinement du trafic réseau consiste à limiter le trafic réseau circulant entre les sites de Fog. Cette caractéristique peut se détailler en trois éléments. Premièrement, le trafic doit être limité aux actions initiées par les utilisateurs : lorsqu'aucun utilisateur ne sollicite le système, il ne doit pas y avoir d'échanges entre les sites. Deuxièmement, lorsqu'un utilisateur accède à des données stockées localement, cela ne doit pas non plus générer de trafic réseau vers les autres sites. Si un site a une activité plus élevée, les performances des autres sites ne doivent pas être impactées. Enfin, lorsqu'un utilisateur accède à des données qui ne sont pas stockées localement sur le site de Fog auquel il est connecté, les données doivent être rapatriées. Les sites qui ne sont pas concernés par cet échange de données ne doivent pas être sollicités.

La topologie physique du réseau doit également être prise en compte : lorsque deux sites échangent des données, il semble préférable que ces échanges concernent des sites

physiquement proches et interconnectés par un lien réseau de faible distance plutôt que deux sites éloignés, sollicitant un nombre important d'équipements intermédiaires.

3.1.4 Fonctionnement en mode déconnecté

La possibilité d'accéder aux données localement en cas de partitionnement du réseau est une caractéristique essentielle. En cas de panne des liens réseaux entre les sites de Fog, les données stockées localement sur le site auquel l'utilisateur est connecté doivent rester accessibles. Les sites doivent donc être les plus indépendants possibles les uns des autres.

3.1.5 Support de la mobilité

Le support de la mobilité consiste pour les utilisateurs à pouvoir se déplacer et par conséquent à changer le site auquel ils sont connectés. Lorsque la donnée accédée n'est pas présente localement, celle-ci doit être rapatriée et relocalisée de manière transparente sur le site courant afin de réduire les temps d'accès des futurs accès.

3.1.6 Passage à l'échelle

Enfin, le passage à l'échelle consiste, lorsque la quantité de données stockées augmente, à éviter que les performances se dégradent. De la même façon, le système doit pouvoir supporter à la fois un grand nombre de sites, un grand nombre de clients et être capable de stocker de grandes quantités de données. Dans ce travail, nous ne traitons pas le problème de savoir à quel site de Fog un client doit-il être rattaché. Nous considérons que le site le plus prêt en matière de latence est celui utilisé. Certains travaux étudient la possibilité de vérifier les contraintes, notamment si les ressources fournies sont suffisantes pour le client avant de réaliser l'affectation [Fan et al. 2015]. Nous ne traitons pas non plus le problème consistant à déterminer si une requête doit être envoyée dans le Fog, traitée en local ou envoyée directement dans une infrastructure de Cloud Computing. Cette question a été traitée dans plusieurs travaux [Yousefpour et al. 2017 ; Olaniyan et al. 2018].

3.2 Modèles de charge

Une des questions principales lorsqu'un système de stockage est conçu est de savoir si celui-ci va pouvoir répondre au besoin, c'est-à-dire, qu'il pourra faire face au nombre et à la fréquence des requêtes et s'il pourra stocker la quantité de données souhaitée. Il n'existe pas à notre connaissance d'article mentionnant le modèle de charge dans une infrastructure Fog : quelle est la taille des données à écrire, à lire ainsi que leur nombre et leur fréquence d'accès. Plusieurs auteurs [Anwar et al. 2016 ; Atikoglu et al. 2012] proposent cependant des exemples de charges pour des systèmes de stockage clés/valeurs dans un contexte de Cloud Computing. Pour Anwar *et al.*, les petits objets ayant une taille d'environ 128 Ko et qui sont très souvent accédés en lecture peuvent correspondre à des pages web tandis que les jeux vidéos en ligne nécessitent des accès en écriture plus

fréquents. Les objets de plus grande taille (128 Mo), les cas d'usage proposés sont le partage de vidéo et la sauvegarde.

3.3 Adéquations des systèmes existants au modèle de Fog Computing

Après avoir listé les différentes caractéristiques que nous souhaitons pour une solution de stockage dans un environnement de Fog Computing, nous allons essayer de vérifier de façon théorique si les solutions existantes répondent aux différents besoins.

3.3.1 Solutions issues des grappes et les grilles de calcul

Les solutions de stockage comme PVFS [Carns et al. 2000], Lustre [Donovan et al. 2003], ou encore RozoFS [Pertin et al. 2014] reposent sur un serveur de métadonnées centralisé. En plus de l'inconvénient au niveau des performances, accéder à un serveur de métadonnées distant ne permet pas de respecter, ni le critère de confinement du trafic réseau ni le critère du fonctionnement en mode déconnecté. En d'autres mots, pour accéder à une donnée stockée localement, il faut d'abord joindre un site distant pour localiser cet objet, générant du trafic inter-sites. De même, lorsque le serveur de métadonnées n'est pas accessible, il devient alors impossible d'accéder aux données stockées localement sur le site de l'utilisateur (pas d'accès aux données lorsque le réseau est partitionné).

De ce fait, les solutions de stockage développées pour les grappes de calcul et les grilles ne semblent pas adapté aux environnements de Fog Computing.

3.3.2 Solutions issues des infrastructures de Cloud Computing

Nous allons maintenant vérifier de façon théorique, si les solutions de stockage utilisées dans les infrastructures d'informatique en nuage sont adaptées au Fog Computing.

Contrairement aux solutions de stockage pour les grappes et les grilles de calcul, ces systèmes ne s'appuient pas sur un serveur de métadonnées centralisé.

Par exemple, Rados [Weil et al. 2007] utilise la fonction de placement CRUSH (*Controlled Replication Under Scalable Hashing*) [Weil et al. 2006b]. Cette fonction permet de déterminer la localisation d'un objet à partir de son nom, de la topologie du réseau ainsi qu'à partir de contraintes et de règles de placement définies par l'utilisateur, permettant de garantir la localité des données. Bien que cette approche permette de localiser les données sans avoir besoin d'émettre de requêtes sur le réseau (confinement du trafic réseau), la distribution, de façon consistante, de la topologie du réseau à chaque nœud est une vraie difficulté. Pour cela Rados s'appuie sur le protocole Paxos [Lamport 2002] permettant d'élire un nœud qui sera chargé de distribuer sa version de la topologie. L'inconvénient du protocole Paxos est qu'une majorité de nœuds doivent être contactés pour que l'élection ait lieu, empêchant de fait, le système de fonctionner lorsque le réseau est partitionné.

Les solutions comme Dynamo [DeCandia et al. 2007], Riak [Klophaus 2010] ou encore CouchDB [Anderson et al. 2010] dans sa seconde version, reposent sur un protocole de *gossip*. Les nœuds s'échangent par voisinage la liste des objets qu'ils stockent. De cette façon chaque nœud construit une table dans laquelle est associée chaque objet avec sa localisation. De ce fait, tous les nœuds peuvent localiser les données de façon autonome, même en cas de partitionnement du réseau. Bien qu'accéder à un objet ne requiert pas de solliciter le réseau, l'écriture de nouveaux objets implique que l'ensemble des nœuds doit être contacté avant que ceux-ci puissent être localisables en tout point du réseau. En d'autres mots, Dynamo permet de sélectionner précisément le nœud stockant les données mais ne permet pas de confiner le trafic réseau.

Cassandra [Lakshman et al. 2010] propose une amélioration de Dynamo [DeCandia et al. 2007] en combinant une fonction de hachage et un protocole par *gossip*. Le placement se fait en appliquant la fonction de hachage sur le nom de l'objet. Le résultat de cette fonction donne une valeur qu'il faut ensuite associer à un nœud. Chaque nœud est associé à un intervalle de valeur et stocke les objets pour lesquels la fonction de hachage retourne une valeur comprise dans celui-ci. L'intervalle géré par chaque nœud est propagé par voisinage grâce à un protocole de *gossip*. En conséquence, tous les nœuds connaissent l'intervalle de valeurs géré par chaque serveur du réseau. Contrairement à Dynamo, cette stratégie permet à Cassandra de confiner le trafic réseau. Cependant, l'introduction d'une fonction de hachage ne permet plus de choisir spécifiquement le nœud sur lequel les données sont écrites.

3.3.3 Solutions issues des réseaux de distribution de contenus (CDN)

Pour plusieurs auteurs, la hiérarchie de Fog peut être vue et considérée comme un réseau de distribution inversé [Satyanarayanan et al. 2015 ; Bittencourt et al. 2017 ; Varshney et al. 2017]. Les données émises par les clients sont remontées au travers de la hiérarchie de Fog pour finalement atteindre l'infrastructure de Cloud Computing située à la racine. Toutefois le Fog peut également être utilisé pour mettre en cache des données près des utilisateurs et en ce sens être un système de distribution de contenus [Mahmud et al. 2018]. Dans d'autres travaux, un système de stockage en bordure du réseau, que ce soit de l'Edge Computing ou du Fog Computing est vu comme un réseau de distribution de contenus. Par exemple, pour Yang *et al.* [Yang et al. 2010], les serveurs de stockage à la périphérie du réseau sont utilisés pour mettre en cache des données stockées dans une infrastructure de Cloud Computing. L'approche est identique à une approche de distribution de contenus. Pour nous, la différence importante entre un réseau de diffusion de contenus et le Fog est que dans un réseau de diffusion, les données sont émises du cœur de réseau vers le bord du réseau tandis que dans le Fog, les données sont émises depuis un bord du réseau. Les approches par réseau de distribution de contenus sont conçues pour placer les données près des utilisateurs et permettent donc une localité des données. De même, les approches par CDN créent dynamiquement des réplicas sur les serveurs proches des utilisateurs, permettant un support natif de la mobilité. En revanche, le confinement du trafic réseau va dépendre de la stratégie, et du protocole utilisé pour localiser les données.

3.4 Conclusions

Dans ce chapitre, nous avons d'abord présenté quelles sont les contraintes pour un système de stockage pour les infrastructures de type Fog Computing puis nous avons présenté de façon théorique les éléments qui font qu'un système de stockage respecte ou non les caractéristiques que nous souhaitons. Il apparaît que les approches reposant sur un protocole par inondation permettent de confiner le trafic réseau tandis que les approches de type CDN permettent un support natif de la mobilité.

Conclusion de la première partie

Nous avons vu que le Fog Computing est une alternative au Cloud Computing permettant de fournir à l'Internet des Objets, des ressources de calcul et de stockage à faible latence. De nombreux cas d'utilisation ont été proposés mais tous ont en commun, soit de chercher à rapatrier vers le Cloud des données générées à la périphérie du réseau ou bien de ramener à cet endroit une décision prise de façon centralisée dans le Cloud.

Nous avons établi une liste des différentes solutions de stockage existantes, des systèmes de fichiers centralisés ou distribués, en considérant notamment les solutions de stockage par objets utilisées dans les infrastructures de Cloud Computing. Enfin, nous avons présenté les approches utilisées dans les réseaux de distribution de contenus, afin de placer les réplicas au plus proche des utilisateurs, pour réduire les temps d'accès et éviter qu'un serveur central ne soit saturé.

Nous avons ensuite listé les caractéristiques que nous pensons nécessaires pour qu'une solution de stockage puisse fonctionner dans un environnement de Fog Computing. Nous avons montré que le stockage de données dans les architectures de Fog Computing, réparties sur plusieurs sites est confronté à de nombreux défis qui devront être résolues : comme la localité des données, le confinement du trafic réseau, le support d'un mode déconnecté, de la mobilité des utilisateurs ainsi que le passage à l'échelle. Enfin, nous avons montré de manière théorique qu'aucune des solutions existantes ne satisfaisait ces contraintes.

Dans la suite de ce travail, nous évaluerons de manière expérimentale ces solutions existantes déployées dans un environnement de Fog Computing. L'objectif sera notamment de valider l'analyse conceptuelle que nous venons de présenter. Nous montrerons que le processus de localisation des données est quelque chose de réellement important et que c'est sur lui que repose notamment le respect du confinement du trafic réseau et du fonctionnement en mode déconnecté, lorsque le réseau est partitionné.

Dans la partie suivante, discutons tout d'abord d'une évaluation expérimentale des solutions Rados, Cassandra qui semblent les solutions les plus prometteuses pour un environnement de Fog Computing. Nous évaluons également la solution InterPlanetary FileSystem (IPFS) qui se rapproche d'un réseau de distribution de contenus. Nous proposerons ensuite plusieurs améliorations. La première consiste à ajouter un *Scale-Out NAS*, déployé localement sur chaque site afin de limiter le trafic inter-sites lorsque les données accédées sont stockées sur le site local. La seconde, quant à elle, consiste à modifier la façon de stocker la localisation des données selon la topologie physique du réseau, de sorte à confiner, le plus possible les trafics lors d'accès distants. Enfin, nous proposerons une évaluation expérimentale de ces différentes approches.

Contributions

Introduction de la seconde partie

Dans cette seconde partie, nous présentons nos différentes contributions. Le premier chapitre est consacré à l'évaluation expérimentale des solutions de stockage retenues précédemment, à savoir Rados et Cassandra mais également la solution de stockage Interplanetary FileSystem (IPFS), une solution de stockage qui se comporte comme un réseau de distribution de contenus et que nous présenterons dans la Section 4.1.3. Nous montrons, après avoir mis en évidence les problèmes de passage à l'échelle de Rados, que si l'approche utilisée par Cassandra permet de limiter le trafic réseau, ce système de stockage ne permet pas d'obtenir de faibles temps d'accès, notamment dans un contexte où les liens réseaux entre les sites ont une forte latence. Enfin, nous montrons qu'IPFS, non seulement permet d'obtenir de faible temps d'accès mais également qu'il résiste relativement bien aux variations de la latence réseau inter-sites.

Malgré ses bonnes performances, IPFS possède un inconvénient majeur : il repose sur l'utilisation d'une table de hachage distribuée pour localiser les objets. Cela génère à la fois un important trafic réseau entre les sites, impactant les temps d'accès et empêchant un fonctionnement lorsque le réseau est partitionné.

Les contributions suivantes ont donc porté sur des améliorations dans le protocole IPFS. Nous présentons ces améliorations tout d'abord sur le plan théorique puis pratique. La première contribution consiste à limiter le trafic inter-sites lorsque les objets accédés sont stockés localement sur le site de l'utilisateur. Pour cela, nous proposons de coupler IPFS à un *Scale-Out NAS* déployé indépendamment sur chaque site. L'idée est de mettre à disposition de tous les nœuds d'un site, l'ensemble des objets stockés sur celui-ci. De cette façon la table de hachage n'est pas sollicitée lorsque l'objet demandé est trouvé sur le site. Enfin, la dernière section présente une approche permettant de limiter les trafics réseau inter-sites lorsque l'objet accédé n'est pas stocké localement. Nous proposons dans cette section de remplacer la table de hachage distribuée par un protocole reposant sur un arbre construit sur la topologie physique. Le protocole consiste à rechercher la localisation de la donnée en remontant l'arbre progressivement. Cela permet dans le pire des cas de localiser un objet en contactant le nœud racine, physiquement situé au centre du réseau. De plus, de nouveaux réplicas de localisation sont créés dynamiquement à chaque fois qu'une donnée est répliquée sur un site, permettant aux utilisateurs de localiser ces nouveaux réplicas situé plus proche d'eux avec un faible nombre de sauts. Cela permet de limiter le trafic réseau mais aussi de le confiner, en essayant le plus possible, de ne contacter que des sites proches du site courant. Nous proposons également un algorithme

s'appuyant sur celui de Dijkstra afin de générer l'arbre. La génération de l'arbre est une opération délicate qui doit prendre en compte la façon dont les requêtes sont émises dans notre protocole. Bien qu'un arbre avec une faible profondeur permet de minimiser les latences, cela entraîne une surcharge du nœud racine qui est directement contacté. De même, cela ne permet pas non plus de bénéficier de la relocalisation des enregistrements de localisation. Générer un arbre qui réduit la latence mais qui permet de bénéficier de la relocalisation de réplicas est une véritable difficulté.

Le dernier chapitre de cette partie présente les résultats expérimentaux, obtenus sur la plateforme d'évaluation Grid'5000. Nous montrons de façon pratique que coupler IPFS à un *Scale-Out NAS* permet de réduire les temps d'accès aux données stockées localement. Dans un seconde temps, nous montrons les intérêts de notre approche utilisant un arbre pour la localisation des données. Nous évaluons notre approche successivement sur une micro-comparaison, afin de comprendre en détail le comportement de notre protocole en fonction de l'arbre utilisé, puis sur une macro-comparaison afin de montrer qu'il passe à l'échelle et peut être utilisé sur une réelle topologie. Enfin, nous terminons en essayant d'effectuer une expérimentation dans un environnement plus réaliste. Nous essayons de coupler les plateformes d'expérimentations FIT/IoT-lab et Grid'5000. Malgré les difficultés d'interconnexions et un résultat anecdotique, nous montrons la faisabilité d'une telle approche.

4

Comparaison de solutions existantes en matières de temps d'accès de trafic réseau

Sommaire

Notre première contribution consiste à évaluer si les systèmes de stockage existants peuvent convenir dans un système de Fog Computing. Après une présentation de ces solutions, nous proposons une évaluation des performances de ces dernières sur la plateforme Grid'5000. Nous montrons qu'aucune solution n'est parfaite mais qu'IPFS est la plus prometteuse.

Le travail de ce chapitre a été présenté dans un article du journal Transactions on Large-Scale Data and Knowledge-Centered Systems (TLDKS) [Confais et al. 2017d].

4.1 Configuration et déploiement de solutions de stockage par objets dans un environnement de Fog Computing

Avant de développer notre propre système de stockage pour le Fog, nous cherchons d'abord à vérifier si les systèmes existants peuvent convenir dans un tel environnement. Nous avons retenu trois solutions de stockage, à savoir *(i)* Rados [Weil et al. 2007], *(ii)* Cassandra [Lakshman et al. 2010] qui ont été brièvement présentés dans la partie précédente mais également *(iii)* Interplanetary FileSystem (IPFS) [Benet 2014]. Nous allons présenter les adaptations que nous avons dûes mettre en place pour les déployer dans un environnement de Fog Computing.

4.1.1 Rados

Rados est constitué de trois éléments :

des moniteurs qui permettent de maintenir la topologie logique du réseau (appelée *clustermap*) de façon consistante entre les différents serveurs de stockage et les clients. Cet arbre décrit l'organisation physique des serveurs de stockage et les contraintes de placement. Un moniteur maître est élu grâce au protocole Paxos [Lamport 2002] et se charge de distribuer cet arbre et ces règles aux clients et aux serveurs de stockage ;

des clients qui exécutent la fonction de placement et interrogent directement les serveurs de stockage correspondants ;

des serveurs de stockage appelés OSD (*Objet-based Storage Device* ou *Object Storage Daemon*), qui stockent les données, reçoivent et répondent aux requêtes de clients mais également qui surveillent leurs voisins et remontent tout changement d'état aux moniteurs afin de mettre à jour la topologie distribuée par les moniteurs.

Dans Rados, les données ne sont pas placées directement avec la fonction de hachage CRUSH. Les données sont réparties au sein de « groupes de placement ». Chaque groupe de placement correspond aux données dont les réplicas sont placés exactement sur les mêmes serveurs.

Le placement d'un objet se fait en deux étapes. Premièrement, un hash simple est calculé sur l'identifiant de l'objet. Ce hash permet d'associer l'objet à un groupe de placement. Deuxièmement, la fonction de placement CRUSH est utilisée pour déterminer les serveurs en charge des réplicas des objets stockés dans ce groupe de placement. Cette façon de faire permet d'améliorer les performances du système en pré-calculant le placement des groupes puisque la fonction CRUSH est coûteuse à exécuter. Cette stratégie est illustrée par la Figure 4.1.

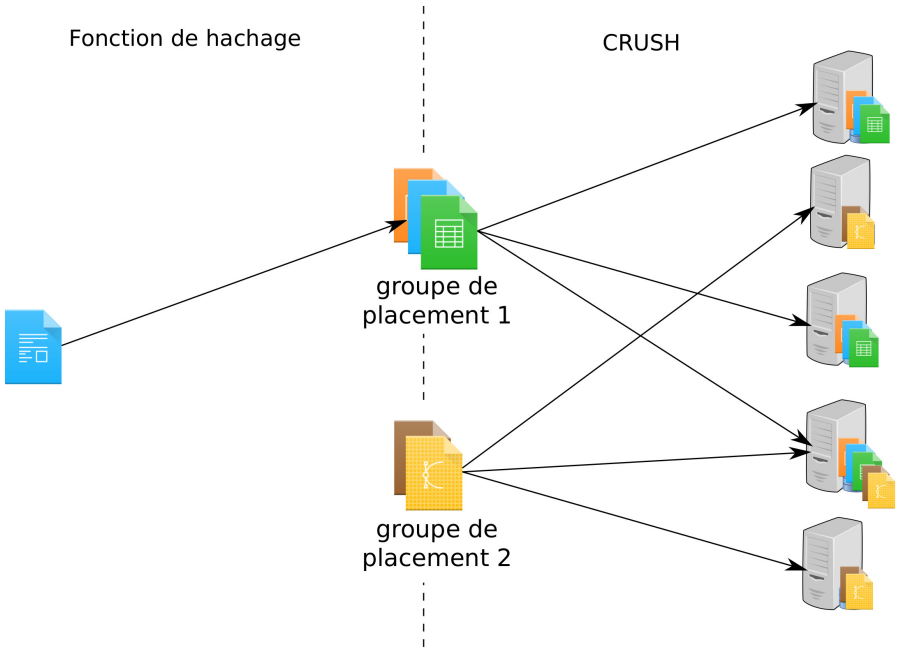


FIGURE 4.1 – Processus de placement utilisé par Rados.

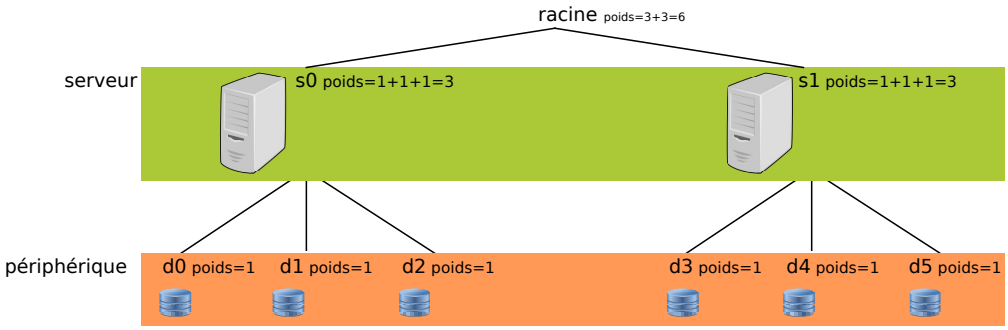
La fonction de placement CRUSH [WEIL et al. 2006b] s'appuie sur des fonctions de hachage, un arbre décrivant l'organisation physique des serveurs de stockage ainsi que sur une liste d'instructions permettant de réaliser le placement. C'est une amélioration des fonctions de la famille RUSH [KARGER et al. 1997] qui ne permettent pas de spécifier des contraintes de placement.

La Figure 4.2 montre un exemple d'arbre et d'une liste d'instructions. Dans l'arbre, les périphériques de stockage situés aux feuilles de l'arbre sont regroupés en serveurs, eux-mêmes regroupés en racks, et enfin par centre de données.

L'idée est d'exécuter la liste d'instructions en parcourant l'arbre depuis la racine et de retourner une liste de périphériques.

Dans l'exemple donné, l'instruction « sélectionner(racine) » permet de sélectionner le nœud racine. L'instruction « choisir(2, serveur) » sélectionne 2 nœuds fils de la racine sélectionnée précédemment. Les nœuds fils sont choisis grâce à une fonction de hachage appliquée sur l'identifiant de la donnée à placer. L'instruction « choisir(1, périphérique) » applique pour chacun des serveurs sélectionnés précédemment, une fonction de hachage permettant de sélectionner un nœud fils. Enfin l'instruction « fin » permet de retourner la liste des périphériques sélectionnés.

Dans l'exemple donné, la règle de placement permet de garantir que les deux périphériques sélectionnés n'appartiennent pas au même serveur. Des règles similaires peuvent être écrites pour garantir un placement des réplicas sur des sites distincts ou sur des racks de serveurs différents. Il est même possible de forcer l'écriture sur un serveur précis et garantir ainsi que l'un des réplicas sera proche des utilisateurs ou encore, construire



(a) – Exemple de clustermap

1. sélectionner(racine)
2. choisir(2, serveur)
3. choisir(1, périphérique)
4. fin

(b) – Exemple de règles de placement

FIGURE 4.2 – Exemple de clustermap et de règle pour le placement de données avec l'algorithme CRUSH.

des règles en fonction de matériels hétérogènes avec des performances variées.

Les fonctions de hachages utilisées peuvent être différentes à chaque nœud de l'arbre. Par exemple la fonction de hachage pour sélectionner un périphérique du serveur $s0$ n'est pas forcément la même que la fonction de hachage permettant de sélectionner un périphérique du serveur $s1$. Nous noterons toutefois que l'algorithme CRUSH permet de réaliser un hachage consistant, c'est-à-dire que la modification de l'arbre entraîne seulement le déplacement des données qui étaient associées à la branche modifiée.

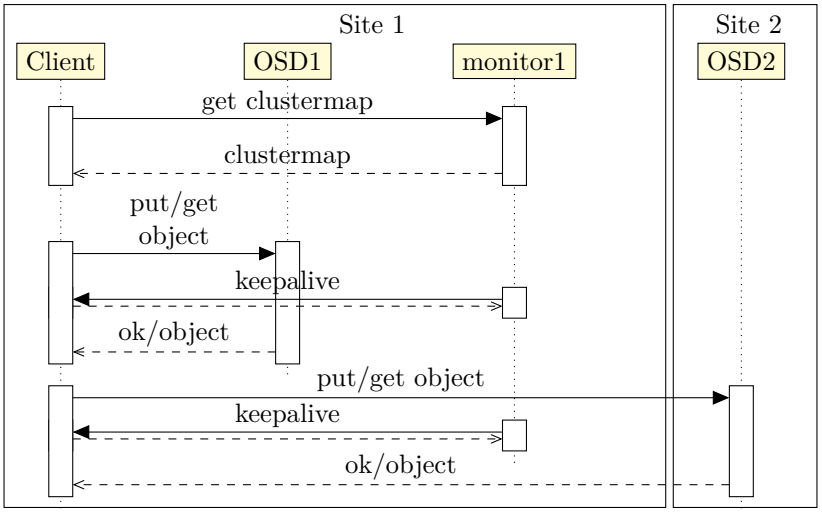
La difficulté de mise en œuvre d'une telle approche est de garantir que tous les nœuds travaillent avec la même version de l'arbre, afin que pour un objet donné, tous les clients soient capables de calculer le même placement. Le rôle des moniteurs est crucial pour cela : à chaque instant, un seul moniteur doit avoir la responsabilité de propager les modifications de l'arbre. L'utilisation d'un seul moniteur garantit cela mais dans un contexte de tolérance aux pannes, il est souvent préférable de déployer plusieurs moniteurs. Tous peuvent transmettre l'arbre aux nœuds de stockage et aux clients, mais seul le moniteur élu en utilisant le protocole Paxos [LAMPORT 2002] est capable de prendre la décision de toute modification.

La Figure 4.3 montre les échanges réseaux que nous observons lorsque Rados est déployé dans un environnement multi-sites, respectivement du point de vue du client (a), d'un serveur de stockage (b) et d'un moniteur (c).

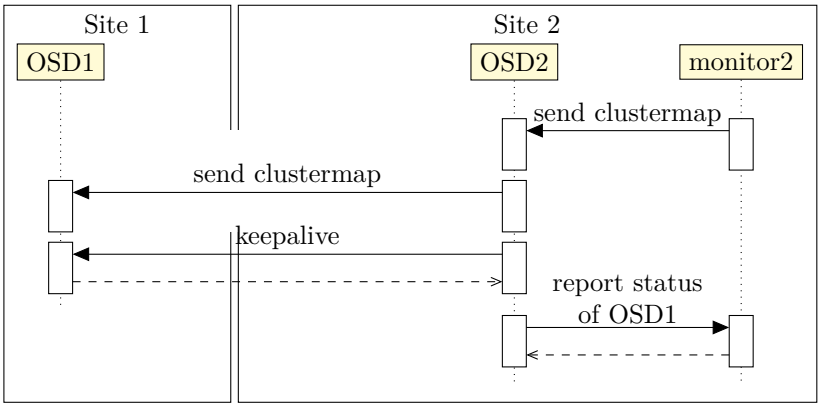
Rados permet aux administrateurs d'organiser les objets au sein de *pools*. Chaque *pool*

est associé à certains paramètres comme un niveau de réplication, des règles de placement qui décrivent les contraintes de placement. Chacun définit un espace de nom. Afin d'écrire ou d'accéder un objet, les clients doivent non seulement connaître l'identifiant de l'objet mais également le nom du *pool* auquel l'objet appartient.

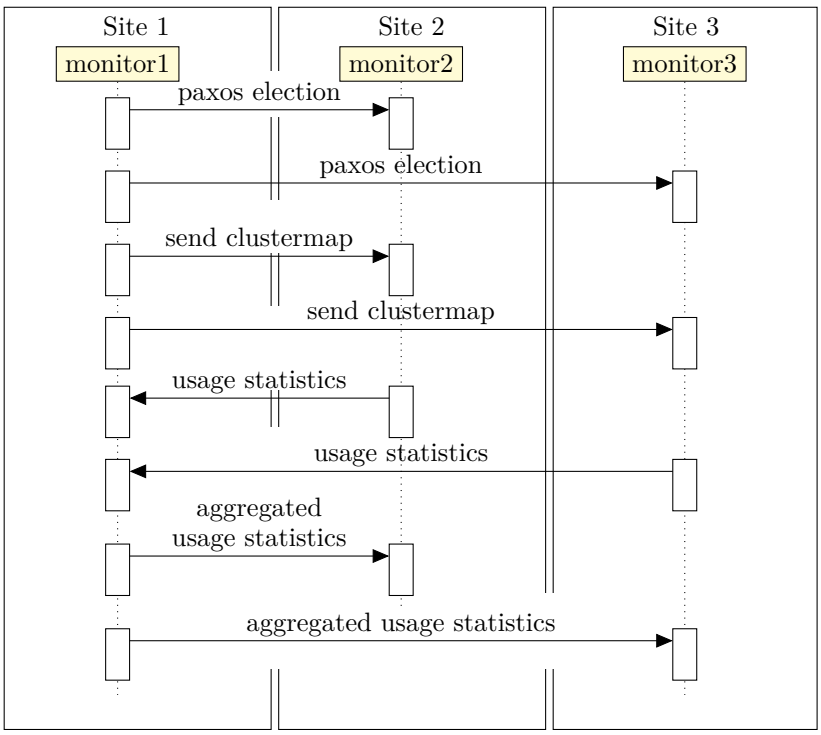
Afin d'introduire la localité des données introduite dans la Section 3.1, nous proposons d'utiliser les règles de placement pour contraindre les objets d'un *pool* donné à être placés sur un site spécifique. La Figure 4.4(a) montre un exemple de *clustermap* décrivant la topologie du réseau dans un environnement de Fog Computing tandis que la Figure 4.4(b) montre les règles de placement associées à chacun des *pools* où l'adéquation entre *pool* et site de Fog est effectuée.



(a) – Du point de vue d'un **client**.



(b) – Du point de vue d'un **serveur de stockage** (OSD).

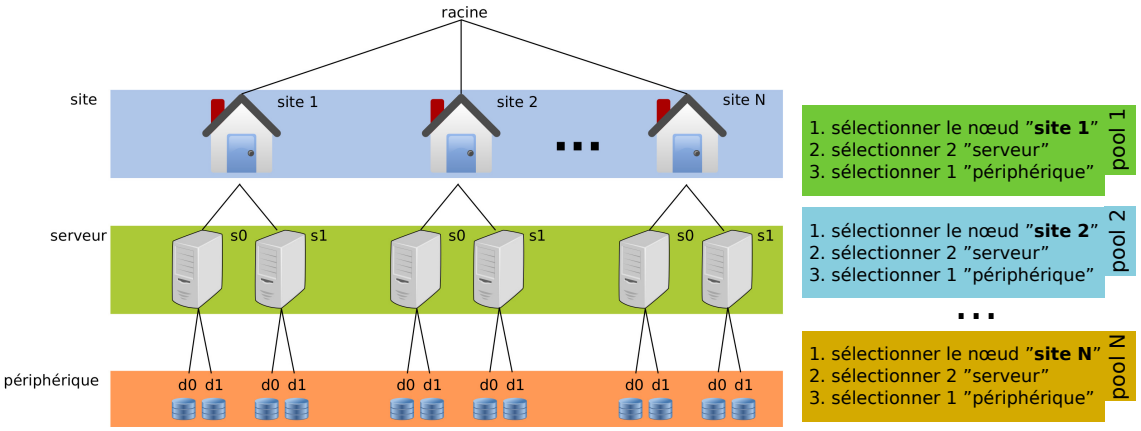


(c) – Du point de vue d'un **moniteur** – le moniteur sur le site 1 est élu, son *clustermap* est utilisé par tous les nœuds.

FIGURE 4.3 – Diagramme de séquence montrant les échanges réseaux observés du point de vue d'un client (a), d'un serveur de stockage (b) et d'un moniteur (c), lorsque Rados est déployé dans un environnement de Fog Computing.

Avec un *pool* associé à chaque site, l'inconvénient est que les données d'un utilisateur ne peuvent pas être déplacées facilement. Si un utilisateur se déplace, ses objets doivent être attribués à un autre *pool*, permettant de placer les objets sur un autre site. De plus, cette approche implique que tous les mouvements de données doivent être initiés par l'utilisateur. Par exemple, si un administrateur prend la décision de déplacer les données de l'utilisateur, ce dernier ne pourra plus les retrouver car il ne connaîtra pas dans quel *pool* celles-ci ont été réaffectées.

Créer un *pool* par utilisateur est une solution à ce problème : lorsque les règles de placements associées à un *pool* sont modifiées, Rados déplace automatiquement les données pour respecter la nouvelle règle. Cette façon de faire permet de faire en sorte que les données suivent l'utilisateur. Toutefois, cette approche pose des problèmes de passage à l'échelle aussitôt que le nombre d'utilisateurs devient conséquent puisque la liste des *pools* et les règles associées sont distribuées à chaque client et serveur de stockage par les moniteurs. De plus, au niveau de la mobilité, cette approche a l'inconvénient de devoir relocaliser et déplacer toutes les données de l'utilisateur à chaque fois que ce dernier change de site. Il n'est pas possible de ne déplacer uniquement les données qui



(a) – Exemple de clustermap

(b) – Exemples de règles de placement

FIGURE 4.4 – Exemples de clustermap (a) ainsi que les règles de placement associées aux pools dans un contexte de Fog (b). Le site stockant les réplicas est précisé dans chaque règle. Par exemple, tous les objets appartenant au « pool N » sont stockés sur le site N.

sont accédées fréquemment. Plus les *pools* contiendront d'objets, plus les déplacements seront coûteux, laissant apparaître un effet ping/pong, où les données sont déplacées sans arrêts entre deux sites donnés.

Une dernière approche serait de déterminer de façon prédictive [TIRADO et al. 2011] depuis quels sites l'objet écrit va être accédé et de le placer dans un *pool* stockant un réplica sur chacun de ces sites. Si l'objet est alors lu depuis un autre site qui n'était pas prévu, il est alors possible de modifier la règle de placement du *pool* pour créer un réplica sur le site voulu. Le problème d'une telle approche est que cela tend à écrire l'ensemble des objets sur l'ensemble des sites.

Sur la Figure 4.3, les trafics réseaux inter-sites sont composés de messages Paxos, de messages de distribution de l'arbre décrivant la topologie du réseau, et de statistiques d'utilisation échangées entre les moniteurs. Pour réduire au plus ce trafic, nous proposons de déployer un moniteur par site. Cela permet de limiter le trafic inter-sites correspondant au report de l'état des serveurs de stockage, mais cela maximise la quantité de trafic correspondant à l'échange de statistiques entre moniteurs.

Nous pouvons utiliser moins de moniteurs mais avoir des sites sans moniteur est la garantie que certains clients devront se connecter à un site distant pour récupérer l'arbre décrivant la topologie du réseau (ou *clustermap*) avant de pouvoir utiliser les ressources du site local. Également, lors d'un partitionnement du réseau, seule la partie possédant une majorité de moniteurs peut continuer de fonctionner. Dans les autres partitions, les clients ne peuvent pas accéder aux objets stockés, faute d'une élection Paxos permettant d'obtenir l'arbre décrivant la topologie du réseau. Enfin pour terminer, nous ne devons pas oublier de préciser que le protocole Paxos utilisé entre les moniteurs est un réel frein au passage à l'échelle de Rados.

Pour conclure, les adaptations principales que nous avons effectuées pour respecter les contraintes du Fog sont :

- Créer un *pool* par utilisateur avec des règles de placements pour contraindre la localité des données ;
- Placer un moniteur par site pour limiter les échanges de métadonnées.

4.1.2 Cassandra

Cassandra [Lakshman et al. 2010] est un système de stockage clé/valeur qui utilise une approche hybride utilisant une approche par inondation (ou *gossip*) et du hachage pour le placement des données.

Comme présenté dans la partie précédente, un espace de clés est réparti entre les nœuds du réseau. Un protocole de *gossip* permet d'informer l'ensemble des nœuds du réseau sur l'intervalle géré par chaque serveur. Enfin, une fonction de hachage permet de déterminer les serveurs stockant un réplica d'un objet donné. Ce fonctionnement est souvent appelé « table de hachage distribuée à un saut » par le fait que chaque nœud peut être contacté directement par n'importe quel autre nœud, sans nécessiter un protocole de routage complexe.

Une particularité de Cassandra est son quorum, permettant de gérer différents niveaux de consistance. Ce quorum, spécifié par les utilisateurs permet de définir le nombre de réplicas qui ont besoin d'être lus ou écrits pour valider une opération. Selon la valeur de ce quorum, Cassandra peut fournir différents niveaux de consistance. Ce quorum fournit un compromis entre les temps d'accès et la consistance.

Les principaux échanges du protocole utilisé par Cassandra sont illustrés dans la Figure 4.5. Les clients reçoivent la topologie lorsqu'ils se connectent au serveur de leur choix. Ils se connectent ensuite à plusieurs nœuds. Plusieurs stratégies existent pour sélectionner à quel serveur les requêtes sont envoyées. Par défaut, les requêtes sont envoyées aux différents serveurs en utilisant une approche par tourniquet (ou *round-robin*). Chaque serveur transfère ensuite la requête au serveur stockant la clé demandée (en utilisant la stratégie décrite précédemment).

Considérations pour le Fog

Cassandra propose d'utiliser plusieurs espaces de clés. Chaque nœud gère alors un intervalle de valeurs différent pour chaque espace de clés. Chaque espace est associé à une stratégie de placement. Par exemple, l'approche « NetworkTopologyStrategy » est une stratégie de placement qui permet de préciser combien de réplicas doivent être stockés dans chaque centre de données (sur chaque site dans notre cas). Afin de garantir la localité des données et réduire les trafics inter-sites, nous décidons de placer uniquement un réplica dans le centre de données local de l'utilisateur. Nous pointons également qu'avoir un seul réplica permet de garantir une consistance forte comme proposée par Rados.

Nous avons exactement le même problème avec les espaces de clés de Cassandra qu'avec les *pools* de Rados. Si un espace de clé par site est créé, l'utilisateur devra se

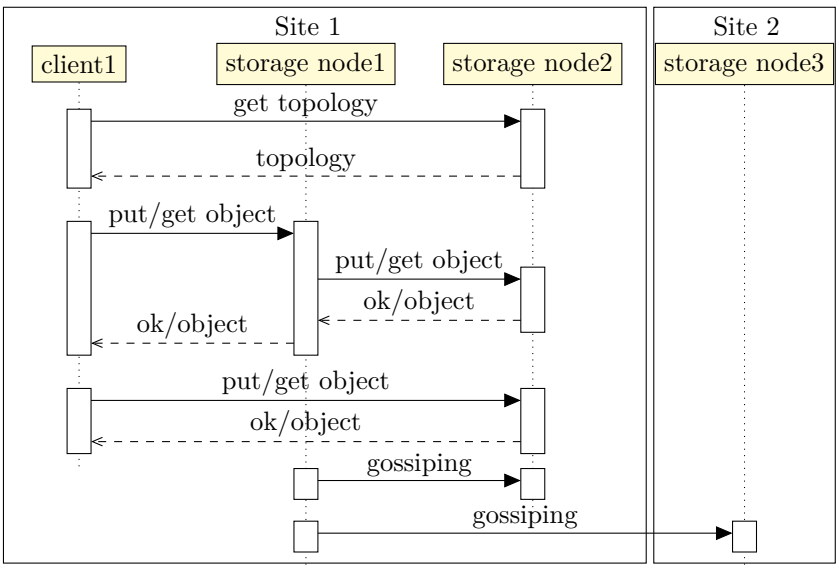


FIGURE 4.5 – Diagramme de séquence montrant les échanges réseaux que nous pouvons observer en utilisant Cassandra dans un environnement de Fog Computing. Le hash de l'objet indique que celui-ci doit être stocké sur le nœud de stockage 2.

souvenir de l'endroit où il a stocké ses données. Avec un espace de clé par utilisateur, les utilisateurs n'ont pas besoin de se souvenir de l'espace de clé associé aux données mais le passage à l'échelle est compromis. Dans Rados, la liste des *pools* est diffusée par les moniteurs. Dans Cassandra, la liste des espaces de clés ainsi que les paramètres de réplication sont échangés lors du gossip.

D'après la Figure 4.5, les messages de gossip forment le seul trafic réseau échangé entre les différents sites. La quantité de données échangées est indépendante de l'activité des sites mais varie grandement selon le nombre de nœuds composant le réseau.

Pour conclure, les principales adaptations que nous avons effectuées pour adapter Cassandra à un environnement de type Fog sont :

- Créer un espace de clés par utilisateur ;
- Utiliser la stratégie de placement « NetworkTopologyStrategy » pour garantir la localité des données.

4.1.3 *Interplanetary FileSystem* (IPFS)

Il n'existe pas de systèmes « *sur étagère* » proposant de déployer un réseau de distribution de contenu. InterPlanetary FileSystem [BENET 2014] est ce qui s'en rapproche le plus, en proposant un système de stockage reposant sur le protocole BitTorrent [LEGOUT et al. 2005] et une table de hachage distribuée de type Kademlia [MAYMOUNKOV et al. 2002], qui sont des protocoles connus pour leur capacité de passage à l'échelle.

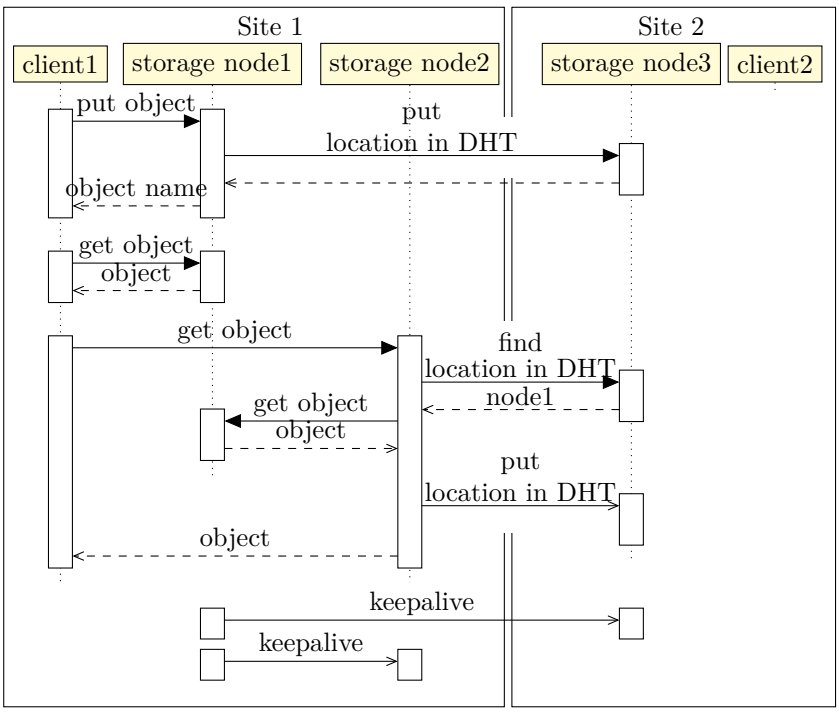


FIGURE 4.6 – Diagramme de séquence montrant les échanges réseaux que nous observons avec IPFS déployé dans un environnement de Fog Computing. Pour simplifier le diagramme, nous ne représentons qu'un seul saut lors de l'accès à la DHT.

Fonctionnement général

Bien que le protocole BitTorrent soit utilisé pour manipuler les objets entre les différents nœuds du système, la table de hachage distribuée Kademlia stocke la localisation des différents objets. Nous insistons sur le fait que la table de hachage distribuée ne stocke que la localisation des objets et non le contenu des objets eux-mêmes. Une telle approche est différente de Rados et Cassandra qui sont conçus pour localiser les objets sans communication supplémentaire. IPFS utilise des objets immuables : la consistance est plus facile à maintenir entre les réplicas mais la modification d'un objet revient à en créer un nouveau. Le protocole BitTorrent permet de télécharger un objet depuis plusieurs sources simultanément [LEGOUT et al. 2005].

Contrairement à Rados et à Cassandra, les utilisateurs ne peuvent pas choisir le nom des objets qu'ils écrivent. Ceux-ci sont générés à partir d'une somme de contrôle calculée sur le contenu de l'objet. Ceci est également une conséquence d'avoir des objets immuables, dans le sens où cela empêche deux utilisateurs d'écrire simultanément sur deux nœuds distincts, deux objets différents portant le même nom.

La Figure 4.6 montre les échanges effectués par IPFS lorsqu'un utilisateur écrit ou lit un objet. Lorsqu'un client veut écrire un objet, il envoie celui-ci au nœud de son choix. Le nœud stocke localement l'objet et ajoute dans la table de hachage distribuée Kademlia,

un enregistrement indiquant qu'il est le nœud stockant un réplica de cet objet.

Réciproquement, lorsqu'un client souhaite accéder à un objet, il contacte le nœud IPFS de son choix. Le nœud vérifie si l'objet souhaité n'est pas stocké localement. Si tel est le cas, alors l'objet est directement transmis au client. Sinon, le nœud IPFS consulte la table de hachage distribuée pour déterminer quels sont les nœuds stockant un réplica de l'objet souhaité. L'objet est ensuite téléchargé à l'aide du protocole BitTorrent avant d'être transmis au client. Une copie de l'objet est également créée sur le nœud interrogé par le client et la table de hachage distribuée Kademlia est mise à jour pour refléter l'existence de ce nouveau réplica.

Cette façon de faire permet de supporter la mobilité de façon native, en relocalisant les données sur les nœuds où elles sont demandées. Un second accès à l'objet depuis ce même nœud peut se faire plus rapidement, en accédant au réplica local, créé lors du premier accès.

Enfin, les nœuds échangent régulièrement des messages avec leurs voisins pour maintenir leur table de routage.

Considérations pour le Fog

Par sa conception, IPFS favorise la localité des objets stockés puisque les objets sont stockés sur le nœud interrogé par le client et nous supposons que le client interroge toujours un nœud du site le plus proche. En revanche, l'utilisation d'une table de hachage distribuée de type Kademlia génère du trafic entre les différents sites.

Pour conclure, nous n'avons pas eu besoin de faire d'adaptations pour qu'IPFS puisse fonctionner dans un environnement de Fog. IPFS peut fonctionner partiellement lorsque le réseau est partitionné : les données sont accessibles aussi longtemps qu'un réplica est accessible et que la localisation puisse être accédée dans la table de hachage distribuée.

4.1.4 Conclusions

Pour conclure, Rados permet la localité des données. Grâce aux règles de placement, il est possible de stocker les données sur le site le plus proche de l'utilisateur. Le confinement du trafic réseau n'est que partiel car l'utilisation du protocole Paxos entre les moniteurs génère du trafic inter-sites non sollicité. L'utilisation du protocole Paxos empêche également Rados de fonctionner en mode déconnecté puisque l'élection Paxos n'a lieu que si au moins la moitié des moniteurs est accessible. Dans les partitions n'atteignant pas ce nombre de moniteurs, aucune élection ne peut se produire et les clients ne sont alors pas capables de télécharger l'arbre décrivant la topologie et donc de localiser et d'accéder aux données. Le support de la mobilité n'est que partiel puisque les objets ne sont déplacés que lorsque les règles de placement sont modifiées manuellement par un administrateur. Enfin le passage à l'échelle est limité par la capacité du protocole Paxos à gérer un grand nombre de nœuds. Nous notons également qu'un déploiement spécifique de Rados dans un environnement multi-sites est proposé, en fédérant plusieurs grappes Rados indépendantes[1]. Toutefois, l'approche proposée ne permet pas de déplacer les objets entre

[1] http://docs.ceph.com/docs/giant/radosgw/federated-config/

les différentes régions et n'est donc pas adapté à une approche de type Fog Computing.

Cassandra, permet comme Rados une localité des données par la spécification de stratégies de placement (que nous expliciterons dans le chapitre suivant). Le trafic réseau inter-sites est confiné puisque seul le trafic du *gossip* est échangé entre les nœuds. La quantité de trafic ne dépend ni de l'utilisation de chaque site, ni du nombre d'objets stockés mais seulement du nombre de nœuds de stockage présents dans le réseau. Étant donné que tous les nœuds connaissent la portion des clés gérées par chaque nœud, il est possible de localiser les objets même lorsque le réseau est partitionné. Les objets stockés dans la partition peuvent tous être accédés au sein de la partition. Nous noterons quand même, que selon le quorum utilisé, certains accès nécessitant d'accéder à un réplica stocké dans une autre partition peuvent ne pas fonctionner. La mobilité des données n'est pas supportée. Enfin, le trafic limité échangé entre les sites permet de supporter à la fois beaucoup de clients et un nombre important de sites.

Enfin, pour IPFS, sont fonctionnement est tel que la localité des données est un critère automatiquement satisfait. Les données sont également automatiquement relocalisées en cas de déplacement de l'utilisateur. En revanche, le confinement du trafic réseau et l'accessibilité aux données lorsque le réseau est partitionné dépend de la table de hachage distribuée qui ne fournit pas de localité.

Cela est résumé dans le Tableau 4.1

	Rados	Cassandra	IPFS
Localité des données	**Oui**	**Oui**	Oui
Confinement du trafic réseau	**Partiellement**	Oui	**Non**
Fonctionnement en mode déconnecté	**Partiellement**	Oui	**Partiellement**
Support de la mobilité	**Partiellement**	**Non**	Nativement
Passage à l'échelle	**Non**	Oui	Oui

TABLE 4.1 – Caractéristiques attendues d'un système de stockage en mode Fog, *a priori* satisfaites pour Rados, Cassandra et IPFS.

4.2 Comparaison expérimentale des différentes solutions

Maintenant que nous avons présenté les différentes solutions potentielles, nous proposons d'en évaluer les performances sur la plateforme de tests Grid'5000 [BALOUEK et al. 2013].

Après avoir discuté des paramètres et du protocole expérimental dans la Section 4.2.1, nous évaluons tout d'abord le confinement du trafic réseau lors d'accès locaux (Section 4.2.2). Dans la Section 4.2.3, nous évaluons les performances des accès distants.

4.2.1 Matériels et méthodes

Dans cette section, nous présentons les paramètres expérimentaux utilisés pour les différentes expérimentations.

Description de la plateforme

Les évaluations ont été effectuées sur la plateforme Grid'5000, en utilisant la grappe « Paravance » localisé à Rennes (Dell PowerEdge, Intel Xeon, 16 cœurs, 128 Go RAM, 10 Gbps Ethernet).

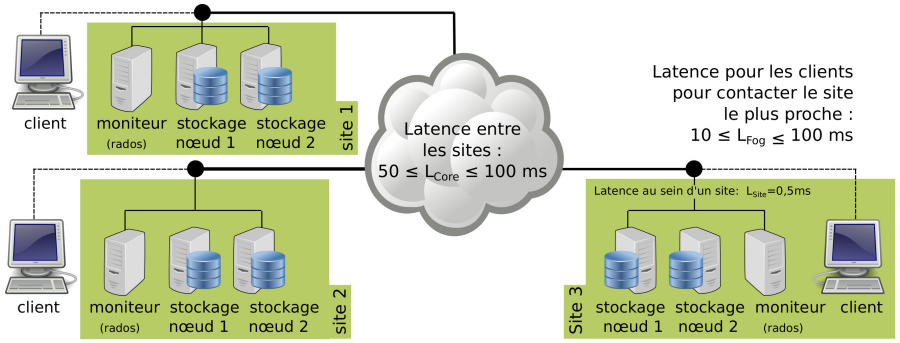


FIGURE 4.7 – Topologie utilisée pour déployer Rados, Cassandra et IPFS, en émulant un environnement de Fog Computing. Les moniteurs ne sont déployés que pour Rados.

Nous émulons l'architecture de Fog présentée dans la Figure 4.7. Chaque site est composé de deux serveurs de stockage. Un serveur supplémentaire est déployé en tant que moniteur lorsque nous évaluons la solution Rados. Finalement, chaque site a son propre client. Le nombre de serveurs de stockage sur chaque site est limité afin de pouvoir réaliser des expérimentations avec un grand nombre de sites tout en gardant l'ensemble des nœuds connectés au même commutateur réseau. Nous sommes ainsi certain que la capacité des liens réseau n'est pas un goulet d'étranglement.

La latence réseau entre les serveurs est émulée artificiellement à l'aide de « Linux traffic control utility » (`tc`). Nous utilisons une latence entre les clients et le site de Fog, $L_{Fog} = 10\,\text{ms}$ et une latence inter-sites de $L_{Core} = 50\,\text{ms}$. La latence entre les serveurs d'un même site est considérée comme faible et est fixée à 0.5 ms. Le débit réseau est spécifié à 10 Gbps, à la fois à l'intérieur des sites et entre les sites de Fog. Plus de détails sur la topologie physique de Grid'5000 sont disponibles à l'adresse suivante : https://www.grid5000.fr/mediawiki/index.php/Rennes:Network.

Nous avons deux raisons, dans ce travail, pour réduire le réseau à ses latences. La première raison est que la latence réseau reflète assez bien la distance géographique. D'après Dabek *et al.* [DABEK et al. 2004b], le délai aller-retour entre deux hôtes dépend fortement de la distance géographique. Comme nous considérons que la distance entre les sites est plus importante que la distance entre chaque client et son site le plus proche, nous avons choisi une valeur de L_{Core} qui est supérieure à notre valeur de L_{Fog}. La seconde raison, est que beaucoup de problèmes réseaux impactent les latences. D'après la loi de Nielsen [NIELSEN 1998] ou des études plus récentes [GETTYS et al. 2012], la latence est également impactée par les phénomènes réseaux comme les congestions, les pertes de paquets, les retransmissions, etc. C'est pourquoi nous ne considérons pas ces phénomènes dans notre évaluation. Enfin, d'après Padhye *et al.*, augmenter la latence a un impact

non négligeable sur le débit atteignable par les connexions TCP [Padhye et al. 1998], un protocole sur lequel repose nos trois solutions.

Afin de garantir la localité des données, nous mettons en place les stratégies de placement pour Rados et Cassandra, présentées respectivement dans les Sections 4.1.1 et 4.1.2. La stratégie de réplication de chaque système est paramétrée de façon à ne stocker qu'un seul réplica de chaque objet. Ce paramètre permet d'obtenir les meilleurs temps possibles pour accéder aux données puisque nous pensons que les mécanismes de réplication ne peuvent que dégrader les temps d'accès et peuvent avoir un impact différent sur les différents systèmes. Désactiver la réplication nous permet également de placer les trois systèmes dans une situation comparable.

Nous avons également modifié le code source d'IPFS pour éviter la réplication dans la table de hachage distribuée. Par défaut la localisation de chaque objet est écrite sur 10 nœuds distincts. Nous avons désactivé ce mécanisme afin de ne stocker qu'une seule fois la localisation de chaque objet. Cela nous permet aussi d'éviter de possibles biais lors de la comparaison des avantages et des inconvénients de chaque système.

Finalement, les métriques que nous relevons sont le temps nécessaire pour effectuer chaque opération de lecture ou d'écriture ainsi que la quantité de trafic réseau échangé entre les sites pendant cette période. Tous les systèmes de fichiers sont démontés puis remontés après chaque accès afin d'éviter les effets de cache entre les exécutions successives.

Outil de mesure et modèle de charge considéré

Nous avons utilisé l'outil *Yahoo Cloud System Benchmark* (YCSB) [Cooper et al. 2010] pour évaluer les trois systèmes. Cet outil est conçu pour évaluer des solutions de stockage par objets dans un environnement de Cloud Computing mais nous n'y trouvons aucune limitation pour l'utiliser dans notre environnement particulier de Fog Computing. YCSB propose différentes charges, selon *(i)* la quantité de données écrites, *(ii)* la taille des objets, *(iii)* les proportions de lectures, de mises à jour et de suppressions effectuées mais aussi selon *(iv)* comment les objets subissant les opérations sont choisis. Pour notre évaluation, nous ne faisons que des écritures et des lectures d'objets de différentes tailles : les clients se connectent au système de stockage, exécutent les opérations d'écriture et de lecture et se déconnectent aussitôt. Aucune requête de mise à jour ou de suppression n'est effectuée et les tailles d'objets ont été choisies pour être représentatives de scénarios réels [Anwar et al. 2016].

Parce que les solutions de stockage peuvent donner de meilleures performances pour certaines tailles d'objets, nous choisissons de les évaluer en utilisant trois tailles d'objets différentes : 256 Ko, 1 Mo et 10 Mo. Les objets de 256 Ko peuvent correspondre aux données d'un jeu en réseau tandis que les objets de 10 Mo correspondent plutôt à des données de sauvegarde. La valeur de 1 Mo correspond à une taille intermédiaire. Le nombre d'objets que nous utilisons varie de 1 à 100 par site, de façon à observer comment le système réagit sous une forte charge.

Le nombre de processus légers utilisé par YCSB est égal au nombre d'objets accédés de façon que tous les accès soient faits en parallèle. Pendant la phase de lecture, chaque objet est lu une et une seule fois grâce à la stratégie d'accès « séquentielle ». Nous n'avons pas

utilisé la stratégie par défaut « Zipfian » qui suit la loi « Zipf », décrivant que peu d'objets sont accédés très souvent et beaucoup sont rarement accédés. Nous n'avons pas non plus utilisé la stratégie « uniforme » car accéder à un même objet plusieurs fois favorise IPFS qui est capable de relocaliser les données. La stratégie d'accès « séquentielle » ne signifie pas que les objets sont lus séquentiellement. Nous considérons les scénarios utilisant 1, 7 et 11 sites. Bien qu'une réelle architecture de Fog puisse être composée de bien plus de sites, nous montrons que 11 sites sont suffisants pour identifier plusieurs difficultés rencontrées par les solutions de stockage dans ce contexte. Nous insistons sur le fait que lorsque les clients écrivent tous le même nombre d'objets sur leur site, le nombre total d'objets stockés dans le système varie selon le nombre de sites puisque ajouter un site revient à ajouter un client.

Chaque expérience est exécutée au moins 10 fois pour obtenir des résultats stables. Les écarts-types ne sont pas précisés car ils correspondent à quelques centièmes de secondes et ne sont pas significatifs. Les temps d'accès correspondent au temps pour écrire ou lire un seul objet (ce temps ne prend pas en compte le temps de connexion et de déconnexion). Nous avons fait ce choix car Cassandra nécessite beaucoup de temps pour établir la connexion avec le système. Enfin, nous avons développé un module d'YCSB écrit en Java permettant d'évaluer IPFS[2]. Ce module de 160 lignes, utilise la bibliothèque *java-ipfs-api* proposée par les développeurs d'IPFS.

Pour prévenir tout biais, le serveur contacté par IPFS pour écrire ou lire chaque objet est sélectionné aléatoirement parmi les serveurs situés sur le site le plus proche. Cela signifie qu'un objet peut être écrit sur l'un des serveurs du site et lu depuis l'autre. Ce comportement est nécessaire afin d'éviter que la table de hachage distribuée ne soit pas consultée. En effet, comme nous l'avons décrit dans la Section 4.1.3, lorsque l'objet est stocké localement sur le serveur interrogé, il est directement envoyé au client, sans consulter la DHT.

4.2.2 Évaluation des accès locaux

Dans ce premier scénario, tous les clients écrivent des objets sur leur site local puis les lisent.

Le but est d'évaluer la localité et le confinement du trafic réseau pour les trois systèmes. Cette dernière propriété est évaluée en mesurant la quantité de trafic réseau échangé entre les sites et en mesurant l'impact de ce trafic au fur et à mesure que le nombre de sites augmente.

Temps d'écriture et de lecture

Les tableaux 4.2(a), 4.2(b) et 4.2(c) montrent respectivement pour Rados, Cassandra et IPFS, les temps moyens pour écrire ou lire un objet. Pour chacun des trois systèmes, les temps d'accès sont du même ordre de grandeur, quel que soit le nombre de sites.

Par exemple, il faut en moyenne $0,96$ secondes par objet lorsque 100 objets de 1 Mo sont écrits sur 1 site et $1,05$ secondes lorsqu'il y a 11 sites. Les valeurs pour Cassandra

[2]Le code source est fourni à l'adresse suivante : https://github.com/bconfais/\acrshort{YCSB}

	Temps moyen d'**écriture** (secondes)				Temps moyen de **lecture** (secondes)			
	Nombre \ Taille	256 Ko	1 Mo	10 Mo	Nombre \ Taille	256 Ko	1 Mo	10 Mo
1 site	1	0,42	0,78	3,40	1	0,39	0,74	2,53
	10	0,35	0,71	3,24	10	0,34	0,64	2,27
	100	0,35	**0,96**	9,45	100	0,32	0,62	**5,83**
7 sites	1	0,44	0,85	3,44	1	0,40	0,77	2,50
	10	0,35	0,68	3,42	10	0,34	0,64	2,34
	100	0,34	1,01	9,41	100	0,32	0,62	6,06
11 sites	1	0,43	0,82	3,74	1	0,40	0,76	2,56
	10	0,36	0,72	3,62	10	0,34	0,65	2,24
	100	0,36	**1,05**	9,50	100	0,32	0,61	**5,80**

(a) – Rados

	Temps moyen d'**écriture** (secondes)				Temps moyen de **lecture** (secondes)			
	Nombre \ Taille	256 Ko	1 Mo	10 Mo	Nombre \ Taille	256 Ko	1 Mo	10 Mo
1 site	1	0,36	0,72	1,74	1	0,34	0,64	1,78
	10	0,21	0,53	1,89	10	0,16	0,46	1,81
	100	0,46	1,26	9,75	100	0,45	**1,10**	8,85
7 sites	1	0,36	0,67	1,92	1	0,30	0,56	1,75
	10	0,22	0,56	2,11	10	0,18	0,42	1,67
	100	0,56	1,28	9,97	100	0,38	0,96	8,89
11 sites	1	0,38	0,67	1,91	1	0,31	0,62	1,80
	10	0,21	0,57	2,06	10	0,17	0,43	1,70
	100	0,55	1,32	9,76	100	0,40	**0,97**	**11,75**

(b) – Cassandra

	Temps moyen d'**écriture** (secondes)				Temps moyen de **lecture** (secondes)			
	Nombre \ Taille	256 Ko	1 Mo	10 Mo	Nombre \ Taille	256 Ko	1 Mo	10 Mo
1 site	1	0,42	0,69	1,69	1	0,23	0,26	0,57
	10	0,24	0,34	1,81	10	0,14	0,25	0,50
	100	0,35	1,23	**12,20**	100	0,22	0,61	**3,95**
7 sites	1	0,41	0,62	1,69	1	0,22	0,36	0,59
	10	0,22	0,43	1,85	10	0,18	0,32	0,51
	100	0,40	1,32	11,54	100	0,25	0,66	3,94
11 sites	1	0,41	0,65	1,65	1	0,26	0,37	0,64
	10	0,24	0,33	1,93	10	0,19	0,26	0,49
	100	0,35	1,16	**11,86**	100	0,22	0,61	**3,93**

(c) – IPFS

TABLE 4.2 – Temps moyens (en secondes) pour écrire ou lire un objet avec Rados (a), Cassandra (b) et IPFS (c) lorsque la topologie comporte 1, 7 et 11 sites. Les valeurs en gras sont particulièrement discutées dans le texte.

suivent la même tendance. En utilisant un mécanisme qui permet de localiser les objets sans accès distants, les temps d'accès pour Rados et Cassandra ne sont pas impactés par le nombre de sites qui composent l'infrastructure. Par exemple pour Rados, il faut en moyenne 5,83 secondes pour lire un objet lorsque 100 objets de 10 Mo sont lus sur un seul site et 5,80 secondes lorsque 11 sites sont utilisés. Avec Cassandra, passer de 1 à 11 sites fait varier les temps d'accès de $1,10$ secondes à $0,97$ secondes lorsque 100 objets de 1 Mo sont lus. Comme nous le verrons, les trafics inter-sites sont envoyés de façon asynchrone et par conséquent, n'ont pas d'impact sur les temps d'accès.

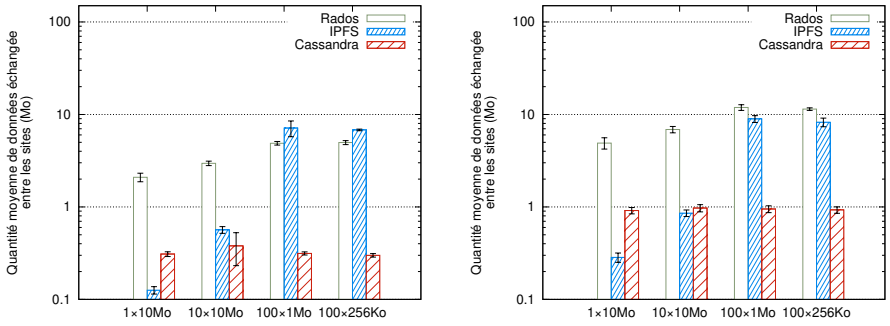
Pour IPFS, les résultats sont surprenants dans le sens où nous nous attendions à voir une dégradation des performances lorsque le nombre de sites augmente. La probabilité de contacter un site distant pour déterminer la localisation d'un objet augmente avec le nombre de sites. Pourtant, nous n'observons pas ce surcoût ($12,20$ vs $11,86$ secondes pour écrire un objet dans un scénario stockant 100×10 Mo par site, avec 3 et 11 sites).

En nous plongeant dans les détails du code, nous découvrons que l'insertion d'un objet se fait en deux étapes. La première consiste à stocker localement l'objet sur le serveur et est effectuée de manière synchrone. La seconde consiste à ajouter la localisation dans la DHT Kademlia. Cette seconde étape est effectuée de manière asynchrone, après que le serveur ait validé l'écriture du client. L'impact de cette seconde étape n'est donc pas visible dans les temps d'accès que nous mesurons.

Nous devrions pourtant le voir sur les temps de lecture, puisque localiser l'objet avant de le récupérer ne peut pas être fait de façon asynchrone. Cependant, le nombre d'objets que nous manipulons n'est pas suffisant pour observer cela ($3,95$ vs $3,93$ secondes pour lire un objet dans un scénario utilisant 100 objets de 10 Mo sur chaque site, avec 3 et 11 sites). De plus, le choix du serveur sur lequel est envoyé la requête est aléatoire et nous n'avons que deux serveurs par site. Cela signifie qu'environ la moitié des requêtes de lecture sont envoyées directement au serveur stockant la donnée, pour lesquelles la DHT n'est pas accédée. Une évaluation spécifique du surcoût de la DHT sera présentée dans le Chapitre 5.

Les tableaux montrent également que pour les trois solutions, les temps d'accès sont plus rapides en lecture qu'en écriture. Il y a plusieurs raisons à cela. La première raison est que les disques durs utilisés pour stocker les données ne fournissent pas les mêmes débits en écriture et en lecture. La seconde raison est qu'avec IPFS, l'écriture nécessite de calculer un hash sur le contenu de l'objet pour déterminer son nom. Ce calcul de hash explique pourquoi les temps d'accès deviennent de plus en plus importants avec la taille des objets : plus un objet est gros, plus le temps d'accès est élevé. Finalement, nous pouvons observer qu'accéder à un grand nombre d'objets en parallèle dégrade les performances ($0,49$ vs $3,93$ secondes pour les charges de 10×10 Mo et 10×100 Mo en lecture avec 11 sites). Dans Rados et Cassandra, le calcul de hash se fait sur le nom de l'objet afin de déterminer son placement. Ce calcul sur le nom n'a donc pas d'impact sur les temps d'accès et nécessite un temps constant, que l'objet fasse 256 Ko ou 10 Mo.

Enfin, nous avons remarqué que dans IPFS, le client a de sérieux problèmes dans la gestion du parallélisme : seulement deux requêtes sont traitées simultanément. Cette mauvaise gestion, mène à dégrader significativement les performances.



(a) – 7 sites

(b) – 11 sites

FIGURE 4.8 – Quantité moyenne de trafic (cumulé) échangé entre les sites lorsque les clients **écrivent** sur leur site. L'échelle est logarithmique et la barre d'erreur représente l'écart-type.

Pour résumer, Rados et Cassandra ont des temps d'accès très stables et seulement la charge de chaque client semble pénaliser les performances. Pour IPFS, nous avons encore besoin d'expérimentations supplémentaires pour quantifier l'impact de la DHT Kademlia sur les temps d'accès. Nous effectuerons cela dans le Chapitre 5.

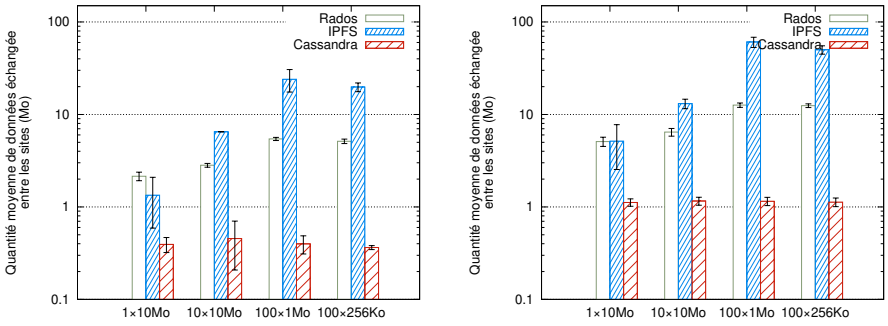
Trafic inter-sites

Les Figures 4.8 et 4.9 montrent la quantité de trafic échangé entre les sites pour les scénarios à 7 et 11 sites.

Premièrement, nous remarquons que cette quantité est vraiment faible au regard de la quantité de données stockées. Cependant, même une faible quantité de données peut avoir un impact important sur les temps d'accès.

Pour Rados, la quantité de trafic échangé entre les sites est similaire en écriture et en lecture. Elle ne dépend que du nombre d'objets manipulés et non de leurs tailles (en augmentant le nombre de sites, nous augmentons le nombre de clients et donc le nombre total d'objets manipulés). Comme décrit précédemment dans la Figure 4.3(c), les nœuds de stockage envoient des notifications et des statistiques aux moniteurs, générant du trafic réseau. Les moniteurs quant à eux envoient du trafic lié au Paxos. Comme discuté dans la Section 4.1.1, il est possible de réduire le nombre de moniteurs pour limiter ce trafic. Toutefois, le coût à payer est que les clients devront alors générer du trafic inter-sites pour récupérer la topologie du réseau et les nœuds de stockage devront reporter leur statut à un moniteur distant.

Pour IPFS, le trafic inter-sites correspond aux messages de la DHT qui sont émis pour localiser les objets et mettre à jour leur localisation. Plus de trafic est échangé lors des lectures puisque lors de la lecture, la DHT est accédée deux fois : une première fois pour localiser l'objet demandé par l'utilisateur et une seconde fois pour mettre à jour la



(a) – 7 sites (b) – 11 sites

FIGURE 4.9 – Quantité moyenne de trafic (cumulé) échangé entre les sites lorsque les clients **lisent** les objets écrits sur leur site. L'échelle est logarithmique et la barre d'erreur représente l'écart-type.

localisation, une fois que le nœud interrogé a relocalisé la donnée (un réplica est créé). Dans le Chapitre 5, nous verrons que ce trafic peut être réduit.

Pour Cassandra, la quantité de trafic croît linéairement avec le nombre de nœuds car chaque seconde, chaque nœud envoie un message de gossip à un autre nœud comme nous l'avons expliqué dans la Section 4.1.2.

Les trafics envoyés de façon asynchrone n'impactent pas les temps d'accès. Si nous calculons la corrélation entre la quantité de trafic émis et le temps d'accès aux données en lecture, pour le scénario utilisant 7 sites, nous trouvons $0,13$ pour Rados, $-0,46$ pour Cassandra et $0,98$ pour IPFS. Cela confirme que pour Rados et Cassandra, la quantité de trafic émis n'impacte pas les temps d'accès. Pour IPFS, le client est en attente lorsque la localisation de l'objet est effectuée et que du trafic est généré, ce qui explique cette corrélation élevée.

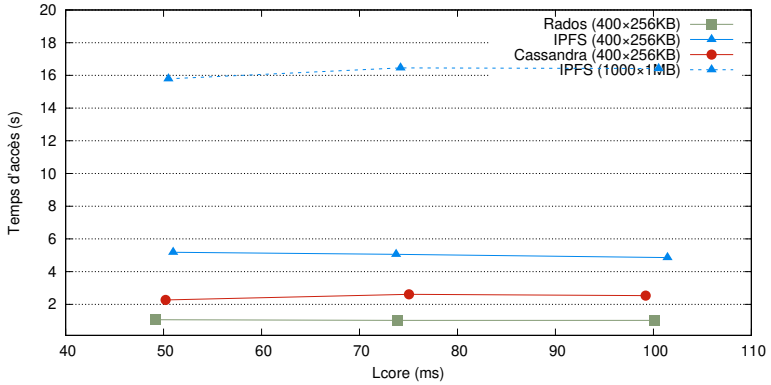
Pour conclure, seulement Cassandra a un bon comportement au regard du trafic réseau échangé lors d'un accès local : la quantité de trafic est inférieure à 1,5 Mo avec 11 sites.

Impact de la latence inter-sites L_{Core}

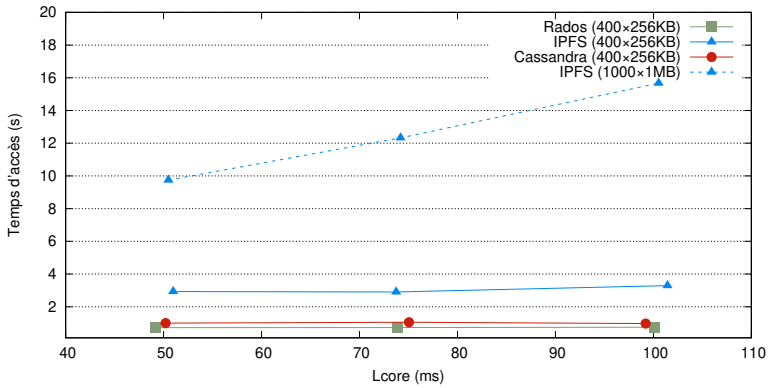
Dans les expérimentations précédentes, nous avons fixé les valeurs arbitraires de latence pour L_{Core} et L_{Fog} respectivement. Pourtant ces latences peuvent avoir un impact non négligeable sur le comportement général des solutions de stockage, et notamment sur les temps d'accès. Nous proposons simplement ici, de faire varier ces latences.

Nous voulons tout d'abord comprendre quel est l'impact de la latence entre les différents sites sur les temps d'accès. En particulier, nous voulons évaluer comment l'accès à la table de hachage distribuée dans IPFS est quelque chose de critique.

Nous fixons la valeur de L_{Fog} à 10 ms comme dans les expérimentations précédentes



(a) – Temps d'écriture moyens (secondes)
en fonction de la latence L_{Core}



(b) – Temps de lecture moyens (secondes)
en fonction de la latence L_{Core}

FIGURE 4.10 – Temps d'accès moyens pour écrire et lire un objet, lorsque la valeur de L_{Core} varie entre 50 et 100 ms et une charge de 400×256 Ko est utilisée sur chacun des 7 sites. Pour IPFS, une charge de 1000×1 Mo est aussi proposée.

et nous utilisons successivement une valeur de 50 ms, 75 ms et 100 ms pour L_{Core}. Nous exécutons le scénario utilisant 7 sites mais nous manipulons 400 objets par site au lieu de 100. La Figure 4.10 le temps moyen pour lire ou écrire un objet en fonction de la latence L_{Core}.

En écriture, la courbe montre qu'augmenter la valeur de la latence entre les sites L_{core} n'a pas d'impact sur les performances. En effet, comme discuté précédemment, il n'y a pas d'échanges synchrones entre les sites lors de cette opération.

Pour la lecture, nous observons qu'IPFS est particulièrement pénalisé d'accéder à la DHT lorsque beaucoup d'objets sont accédés. (la courbe montre un temps allant de $9,74$ s à $15,67$ s selon la valeur de L_{Core}) Nous observerons également que pour Rados, une latence inter-sites élevée et supérieure à 100 ms l'empêche de fonctionner à cause de difficultés à réaliser l'élection Paxos.

Impact de la latence pour accéder au site de Fog L_{Fog}

De la même façon que pour L_{Core}, nous voulons mesurer l'impact de la latence L_{Fog} sur les temps d'accès, c'est-à-dire, l'impact de la latence entre les clients et leur site de Fog le plus proche. La valeur de L_{Core} est fixée à 50 ms tandis que la valeur de L_{Fog} est augmentée de 2 à 100 ms. La latence entre les serveurs d'un même site reste fixée à 0,5 ms.

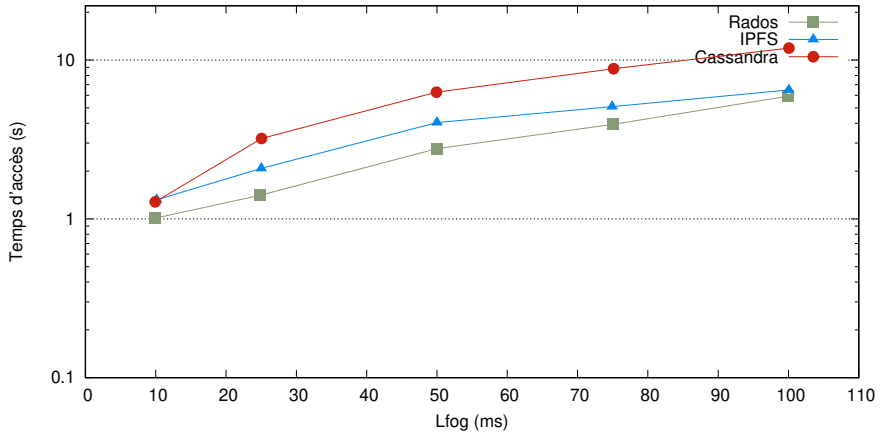


FIGURE 4.11 – Temps d'accès pour écrire ou lire un objet lorsque la latence L_{Fog} varie de 10 à 100 ms. Les valeurs en écriture et en lecture sont les mêmes. Une charge de 100×1 Mo et une topologie de 7 sites est utilisée. Échelle logarithmique sur l'axe y.

La Figure 4.11 montre le temps d'accès moyen à un objet pour Rados, Cassandra et IPFS, en fonction de la latence pour le client, pour atteindre le site de Fog le plus proche. Les tendances sont similaires en écriture et en lecture. Nous ne représentons donc que les valeurs en écriture.

Nous nous attendions à ce que l'impact de la latence L_{Fog} soit le même pour les trois systèmes puisque pour les trois systèmes, le client ne fait qu'envoyer l'objet qu'il veut stocker. Il semble que l'impact de la latence L_{Fog} soit plus important pour Cassandra. Les temps d'accès varient de $1,28$ à $11,91$ secondes ($\times 9,30$) tandis que pour Rados et IPFS, les temps d'accès n'augmentent que d'un facteur 4 (de $1,01$ à $5,02$ s pour Rados et de $1,32$ à $6,49$ s pour IPFS).

La Figure 4.12 montre le protocole utilisé par les clients : lorsqu'un client écrit un objet, il sélectionne un serveur (en utilisant l'algorithme CRUSH pour Rados, une sélection par tourniquet ou *round-robin* dans Cassandra ou de façon aléatoire pour IPFS) et envoie l'objet à celui-ci. Le seul trafic supplémentaire pour Cassandra est lorsque le serveur qui reçoit l'objet doit le transmettre au serveur responsable de la clé. Cependant, ce transfert n'est pas soumis à la latence L_{Fog}. Nous avons également testé la politique *TokenAware* de Cassandra, qui évite ce mécanisme en demandant au client d'écrire directement l'objet sur le serveur responsable de la clé mais les résultats ne sont pas convaincants. Des expérimentations supplémentaires sont nécessaires.

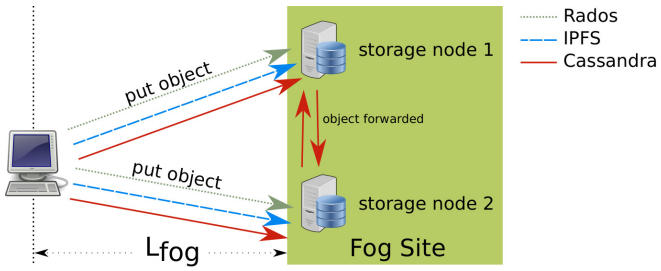


FIGURE 4.12 – Trafic émis lorsqu'un client écrit, quelle que soit la solution de stockage utilisée. Le transfert d'un serveur à l'autre dans Cassandra n'est pas impacté par la latence L_{fog}.

4.2.3 Évaluation des accès distants

La seconde expérimentation que nous effectuons, consiste à évaluer le critère de mobilité. Concrètement, nous voulons analyser quel est l'impact sur le temps d'accès lorsqu'un client accède à une donnée qui n'est pas stockée sur son site local. Nous utilisons la même topologie avec 7 sites que celle présentée précédemment. Rados, Cassandra et IPFS sont déployés de la même façon mais seulement deux clients sont utilisés. Un client pour l'écriture et un pour la lecture. Les autres sites ne servent qu'à fournir des nœuds pour la DHT d'IPFS, des moniteurs pour Rados ou pour générer du trafic de gossip pour Cassandra.

Concrètement, un client écrit sur son site local, les caches sont ensuite vidés et un autre client, situé sur un autre site, lit les données écrites par le premier client. La lecture est effectuée deux fois pour analyser quels sont les avantages de la création automatique de réplicas dans IPFS. Nous rappelons que dans Rados et Cassandra, les objets ne sont pas relocalisés automatiquement sur le site de l'utilisateur lorsque celui-ci y accède. En d'autres mots, les contraintes de placement ne sont pas modifiées dynamiquement dans Rados et Cassandra. Notre but est de montrer le besoin de relocaliser les données près des utilisateurs.

Les Tableaux 4.3(a), 4.3(b) et 4.3(c) montrent les temps d'accès obtenus dans ce scénario avec respectivement Rados, Cassandra et IPFS. Les temps pour écrire les objets sont les mêmes que dans l'expérimentation précédente, lorsque les données ne sont écrites que sur un seul site. Nous avons donc volontairement omis ces valeurs dans le Tableau 4.3.

Pour les lectures (et particulièrement pour la première), le client Rados contacte directement le serveur de stockage (OSD) stockant l'objet sur le site distant (avec une latence réseau de bout en bout égale à $L_{Fog} + L_{Core}$). L'augmentation du temps d'accès est seulement due au transfert de l'objet sur le lien à forte latence entre le client et le serveur distant. Cela prend $9,36\,\text{s}$ de lire un objet de $10\,\text{Mo}$ stocké sur un site distant. Cela est environ cinq fois le temps que nous avions précédemment, lors de la lecture d'un objet local ($2,53\,\text{s}$ dans le Tableau 4.2(a)). Avec Rados, nous observons que la lecture distante est plus performante pour les petits objets. Avec beaucoup d'objets, le parallélisme limite l'augmentation du temps liée à la latence réseau. En effet, le surcoût

Temps moyen en **lecture** (secondes) Première lecture				Temps moyen en **lecture** (secondes) Seconde lecture			
Nombre \ Taille	256 Ko	1 Mo	10 Mo	Nombre \ Taille	256 Ko	1 Mo	10 Mo
1	1,80	3,34	**9,36**	1	1,84	3,42	9,21
10	1,80	3,31	**8,38**	10	1,83	3,29	8,10
100	1,72	3,09	**12,14**	100	1,72	3,09	11,66

(a) – Rados

Temps moyen en **lecture** (secondes) Première lecture				Temps moyen en **lecture** (secondes) Seconde lecture			
Nombre \ Taille	256 Ko	1 Mo	10 Mo	Nombre \ Taille	256 Ko	1 Mo	10 Mo
1	1,46	3,01	**9,65**	1	1,41	3,06	9,43
10	1,53	3,97	12,43	10	1,53	3,75	12,52
100	3,77	8,26	**19,86**	100	3,87	8,14	21,43

(b) – Cassandra

Temps moyen en **lecture** (secondes) Première lecture				Temps moyen en **lecture** (secondes) Seconde lecture			
Nombre \ Taille	256 Ko	1 Mo	10 Mo	Nombre \ Taille	256 Ko	1 Mo	10 Mo
1	0,99	1,24	3,03	1	0,18	0,43	**0,35**
10	0,70	1,15	5,23	10	0,16	0,36	0,31
100	1,28	**4,00**	38,85	100	0,21	0,61	2,96

(c) – IPFS

TABLE 4.3 – Temps moyens (en secondes) pour lire deux fois un même objet stocké sur un site distant, avec Rados (a), Cassandra (b) et IPFS (c). La topologie est composée de 7 sites mais seulement un est utilisé pour réaliser les lectures. Les valeurs en gras sont particulièrement discutées dans le texte.

de temps lié à la latence est réparti sur le nombre d'objets lus en parallèle. Par exemple avec seulement 10 objets de 10 Mo, la lecture d'un objet est en moyenne 4 fois plus longue (8, 38 vs 2, 27 s) mais avec 100 objets, celle-ci ne devient que 2,08 fois plus longue (12, 14 vs 5, 83 s).

Avec IPFS et Cassandra, les requêtes sont envoyées à un serveur de stockage local. C'est ce serveur qui localise et rapatrie l'objet avant de le transmettre au client (comme nous l'avons illustré dans les Figures 4.5 et 4.6). Ce mécanisme augmente les temps de lectures. De plus, nous précisons que dans le test précédent, environ 50 % des requêtes étaient directement envoyées sur le nœud stockant l'objet souhaité. Dans ce scénario, toutes les requêtes doivent être transmises à un serveur situé sur un site distant.

Pour Cassandra, la lecture distante d'un objet de 10 Mo nécessite 9, 65 s tandis que lorsque l'objet était stocké localement, cela ne nécessitait que 1, 78 s (Tableau 4.2(b)). La localisation de l'objet avec Cassandra n'implique pas de communication supplémentaire et n'est donc pas responsable de cette augmentation. Comme pour Rados, l'augmentation du temps est seulement due au transfert de données sur un lien à forte latence. L'augmentation

est plus forte que celle de Rados car Cassandra envoie les données en petits paquets, renforçant l'impact de la latence. Cassandra est le système pour lequel l'augmentation de temps est la plus forte : 19,86 s sont nécessaires pour la lecture d'un objet distant dans un scénario de 100 objets de 10 Mo alors qu'il suffisait de $8,85$ s pour un accès local ($\times 2.25$).

Pour IPFS, lorsque la lecture est effectuée, le nœud local stocke l'objet localement et met à jour la table de hachage distribuée de manière asynchrone. L'augmentation du temps d'accès provient du transfert des données mais également de l'utilisation de la DHT. Dans ce scénario, la DHT est consultée pour localiser tous les objets tandis que lors de la lecture distante, elle ne l'était pas lorsque l'objet demandé était stocké localement. Nous observons, dans tous les cas (sauf pour de gros objets de 10 Mo), que les temps d'accès lors de la lecture distante sont 5 fois plus élevés que lors d'une lecture locale (par exemple $4,00$ vs $1,23$ s dans le Tableau 4.2(c) lorsque 100 objets de 1 Mo sont accédés).

La dernière colonne du Tableau 4.2(c) montre que les temps d'accès lors de la seconde lecture sont du même ordre de grandeur que dans l'expérimentation précédente. Par exemple, $0,35$ secondes sont nécessaires à IPFS pour lire 1 objet de 10 Mo (vs $0,57$ s dans l'expérimentation précédente). L'amélioration des temps d'accès par rapport à l'expérimentation précédente est qu'ici, nous ne sollicitons qu'un seul site alors que la DHT est répartie sur les 7.

Nous devons également préciser que lorsqu'un objet est demandé à un nœud qui ne le stocke pas localement, ce nœud télécharge à la fois le réplica stocké sur le site distant mais aussi le réplica stocké sur le site local, qui fut stocké lors de la première lecture. Les deux réplicas sont accédés en parallèle. Cette stratégie peut dégrader les performances et augmenter la quantité de trafic réseau échangé entre les sites. Pour conclure sur IPFS, les temps d'accès sont faibles parce que le nœud interrogé sert une copie de l'objet qui a été créé lors de la première lecture.

Pour Rados et Cassandra, les temps d'accès sont identiques pour la première et la seconde lecture. La mobilité des données n'est pas implicite et la seconde lecture génère un second accès distant. Comme les temps d'accès sont élevés lors d'un accès distant, nous évaluons dans la partie suivante quel est l'impact de la latence réseau sur ceux-ci.

Impact de la latence inter-sites L_{Core}

Le but de cette expérimentation est de montrer comment la latence inter-sites L_{Core}, impacte les temps d'accès lors d'une lecture distante et de montrer que relocaliser les données sur le site local est quelque chose de nécessaire. Nous effectuons le même test que celui effectué dans la Section 4.2.3 et nous faisons varier la latence L_{Core} entre 50 ms et 100 ms.

La Figure 4.13 montre les temps d'accès que nous obtenons lors d'une lecture distante. La première chose que nous remarquons est que Rados et Cassandra se comportent de la même façon que lorsque nous faisons varier L_{Fog} dans un scénario d'accès locaux puisque les données sont transférées en utilisant les liens inter-sites dont nous faisons varier la latence. Comme nous l'avions montré avec la Figure 4.10 montrant l'impact de la latence inter-site dans un scénario d'accès local, les métadonnées envoyées de façon asynchrones

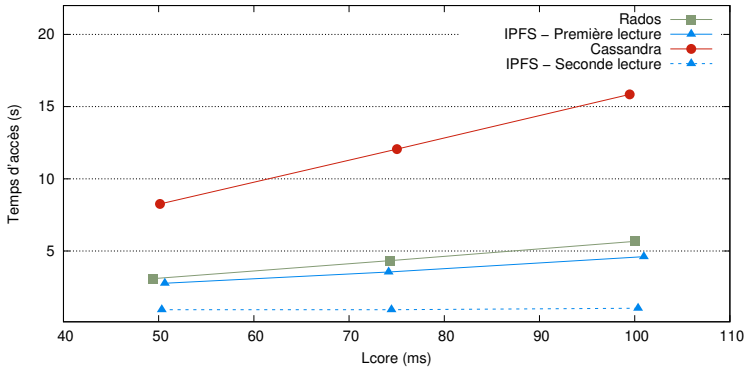


FIGURE 4.13 – Temps d'accès moyen pour lire un objet stocké sur un site distant lorsque la latence L_{core} varie de 50 à 100 ms. Une charge de 400 × 256 Mo et une topologie de 7 sites est utilisée.

n'impactent pas les temps d'accès. Par conséquent, l'augmentation que nous observons ici, correspond simplement aux transferts de données.

Pour IPFS, l'augmentation de la latence inter-sites impacte non seulement les transferts de données mais aussi les performances de la DHT qui permet de localiser les objets. Comme dit précédemment, lors de la seconde lecture, soit l'objet est déjà stocké sur le nœud interrogé et celui-ci est alors envoyé directement au client, soit il ne l'est pas et les deux réplicas existants seront accédés. La légère augmentation des temps d'accès provient que dans ce dernier cas, les deux réplicas ont besoin d'être localisés en utilisant la DHT.

Nous pensons qu'avec 1000 objets, la seconde lecture sera beaucoup plus impactée par la valeur de L_{Core} car l'accès à la DHT représentera une part plus importante de la charge. Cette expérimentation supplémentaire montre que la latence inter-sites a un rôle important lors des accès distants. À la première lecture, les objets devraient être relocalisés sur le site le plus proche de l'utilisateur afin d'éviter le plus possible ce type de lectures.

Surcoût de l'utilisation d'une table de hachage distribuée dans IPFS

Nous terminons par évaluer le surcoût lié à l'utilisation d'une table de hachage distribuée dans IPFS. Nous nous concentrons sur les accès locaux d'un seul site, lorsqu'un client écrit et lit des objets stockés localement.

Cette expérimentation a été effectuée avec une topologie composée de 3 sites. Chaque site contenant 4 serveurs de stockage. L'un des sites est associé à 10 clients, ce qui nous permet de fortement augmenter le nombre d'objets à stocker.

La latence réseau pour atteindre le site de Fog est toujours fixée à $L_{Fog} = 10$ ms mais nous augmentons la latence inter-sites à une valeur extrême de 200 ms. Cette latence peut correspondre à la latence d'un site mobile, situé par exemple dans un train ou un bus. Nous utilisons un `tmpfs` comme système de stockage sous-jacent afin de ne pas être impacté par les performances d'un disque dur ou d'un système de fichiers de type `ext4`.

Enfin, nous implémentons notre propre banc d'essais plutôt que d'utiliser YCSB. Le but est de pouvoir facilement mesurer le temps de chacun des objets individuellement mais aussi d'éviter certains biais. Par exemple, le temps de générer le contenu d'un objet aléatoirement est pris en compte dans le temps d'écriture mesuré par YCSB. Chacun des 10 clients écrit et lit 100 objets sur le site (pour un total de 1000 objets accédés).

La Figure 4.14 montre pour un client donné le temps de lecture de chacun des 100 objets de 256 KB stockés sur le site.

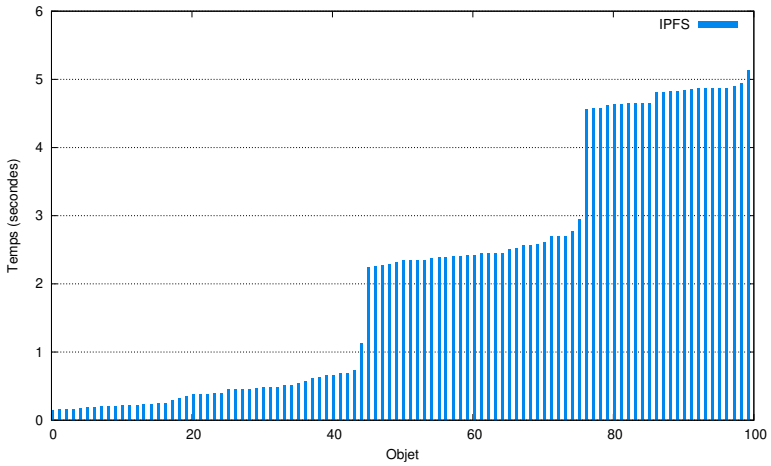


FIGURE 4.14 – Temps pour un client donné de lire chacun des 100 objets de 256 KB. (valeurs pour une itération donnée - les objets sont triés par temps d'accès croissants).

La première chose que nous observons, c'est une grande différence dans les temps d'accès entre le premier objet lu et le dernier. Nous observons 3 plateaux, délimitant des groupes d'objets ayant des temps d'accès similaires. Les 25 premiers objets forment un premier plateau d'objets qui sont lus rapidement. Comme il y a 4 serveurs sur le site, ces requêtes correspondent probablement aux requêtes qui ont été émises au serveur de stockage stockant l'objet demandé. Ces objets n'ont pas eu besoin d'être localisés grâce à la DHT. Le second plateau (objets 25 à 43) montre les objets qui ont été relocalisés sur le nœud interrogé mais pour lesquels la localisation stockée dans la DHT était stockée sur un serveur du site. Par conséquent, les liaisons inter-sites n'ont pas été utilisées, permettant de garder un faible temps d'accès. Enfin, les derniers objets correspondent aux objets pour lesquels la localisation est stockée sur un des deux sites distants et pour lesquels les liens inter-sites ont besoin d'être utilisés.

Cette expérimentation montre qu'avec une grande latence entre les sites de Fog et plus de clients, l'accès d'IPFS à la DHT devient quelque chose de coûteux.

Nous devons également préciser que notre expérimentation utilise un faible nombre de nœuds et que la plupart des accès se font en un seul saut logique. Nous pouvons donc nous attendre à une dégradation plus importante des performances dans un réseau comportant plus de nœuds et où les accès se feront majoritairement en plusieurs sauts logiques.

4.2.4 Résumé de l'évaluation des trois solutions de stockage

Pour conclure, les temps d'accès dépendent essentiellement de 4 paramètres : *(i)* le nombre de sites, *(ii)* les latences pour joindre le site de Fog, *(iii)* la latence inter-sites mais également *(iv)* le nombre d'objets accédés.

Le Tableau 4.4 résume pour chaque solution, quels sont les paramètres qui ont un impact important sur les temps d'accès. Un « ☺ », indique que les temps d'accès ne sont pas ou peu impactés par le paramètre et que la solution se comporte correctement vis-à-vis de ce dernier pour pouvoir être utilisée dans un environnement de Fog Computing. Au contraire, un « ☹ » montre que les temps d'accès sont sensibles à un facteur extérieur, et que des difficultés pourront être rencontrées lors d'un déploiement dans un environnement de Fog Computing.

	Rados	Cassandra	IPFS
Nombre de sites	☹☹	☹☹	☺
Latence L_{fog}	☺	☹☹	☺☺
Latence L_{core}	☹☹	☺	☺
Nombre d'objets accédés	☹	☺☺	☹☹

TABLE 4.4 – Stabilité des temps d'accès lorsque les valeurs des paramètres suivants augmentent.

Rados est sensible à la latence inter-sites (L_{Core}) car avec une latence trop élevée (100 ms), il arrête de fonctionner. Le protocole Paxos et le nombre de « pools » nécessaire dans notre contexte, l'empêche de passer à l'échelle dans un environnement de Fog Computing .

Cassandra est sensible à la latence pour atteindre le site de Fog (L_{Fog}) ainsi qu'à la latence inter-sites (L_{Core}). Ajouter des sites génère plus de trafic réseau. Néanmoins, il diffère des autres solutions parce que la charge, c'est-à-dire la taille et le nombre d'objets accédés, n'a pas d'influence sur les temps d'accès. Cassandra peut passer à l'échelle sur un grand nombre d'objets mais semble limité à un faible nombre de sites, du fait qu'obtenir le statut et les espaces de clés des autres nœuds peut prendre du temps.

Nous avons montré qu'IPFS fournit de faibles temps d'accès, qui ne sont pas beaucoup impactés lorsque les latences L_{Fog} et L_{Core} augmentent (bien que le nombre d'objets doit rester faible pour ne pas solliciter trop fortement la DHT). La seconde expérimentation a montré que la relocalisation des données est quelque chose de nécessaire dans un environnement de Fog computing : IPFS est la seule solution à placer et relocaliser les données proches des utilisateurs de façon native. Pour cette raison, IPFS est peut-être le meilleur candidat pour un système de stockage dans un tel environnement. Toutefois, la quantité de trafic échangé entre les sites est très fortement corrélée avec les temps d'accès ce qui peut être un problème de passage à l'échelle pour stocker une quantité importante de données.

4.3 Conclusions

Ce chapitre a été l'occasion de présenter le fonctionnement de trois solutions de stockage, Rados, Cassandra et IPFS et de montrer comment nous pouvions les déployer dans un environnement de Fog Computing. Nous avons par la suite proposé une évaluation des performances de ces trois systèmes, que ce soit pour mesurer les temps d'accès à un objet stocké localement sur le site de l'utilisateur ou sur un site distant. Nous avons montré que Cassandra a le meilleur comportement possible lorsqu'il s'agit d'accéder à une donnée locale mais que ce système est sensible aux latences intra et inter-sites. Nous avons également montré que Rados ne pouvait pas passer à l'échelle à cause de son utilisation de Paxos. Enfin, nous avons montré qu'IPFS, bien que la table de hachage distribuée soit un frein à une utilisation dans un environnement de type Fog Computing, permettait de relocaliser de façon transparente les données sur les sites où ces derniers sont accédés.

Dans le chapitre suivant, nous proposons de coupler IPFS à une solution de stockage de type *Scale-Out NAS* sur chacun des sites de façon indépendante. Cela doit permettre de ne pas avoir à localiser l'objet en utilisant la table de hachage distribuée lorsque l'objet demandé est stocké sur un autre serveur situé sur le même site.

5

Conception d'un système de stockage distribué pour le Fog

Sommaire

Nous avons montré dans le chapitre précédent que les trafics inter-sites sont coûteux. Il est donc important de chercher à les limiter au maximum, à la fois pour améliorer les temps d'accès et pour permettre au système de fonctionner dans un environnement avec

des latences élevées. Dans l'idéal, les seuls trafics inter-sites devraient être des trafics sollicités, lorsque des objets sont accédés depuis un site distant.

Dans ce chapitre, nous nous concentrons sur le système de stockage IPFS [Benet 2014], sur lequel nous avons obtenu les résultats les plus intéressants lors de nos expérimentations. Nous présentons deux améliorations. La première consiste à coupler IPFS avec une solution de stockage de type *Scale-Out NAS* (*cf* Section 2.1) afin de confiner le trafic au site local lorsque l'objet accédé est stocké localement sur le site. La seconde amélioration consiste à stocker la localisation des données dans un arbre respectant la topologie physique du réseau, de façon à confiner les trafics inter-sites dans le processus de localisation des objets.

5.1 Réduction des trafics réseau inter-sites pour les accès locaux

Après avoir montré que la table de hachage distribuée ne permettait pas de confiner le trafic réseau, nous proposons dans cette section, de coupler IPFS avec un système de fichiers distribué de type *Scale-Out NAS* afin de pouvoir accéder aux objets stockés localement sur le site sans émettre de requêtes à destination d'un autre site.

Ce travail a été présenté à la session 2017 de la conférence ICFEC [Confais et al. 2017b].

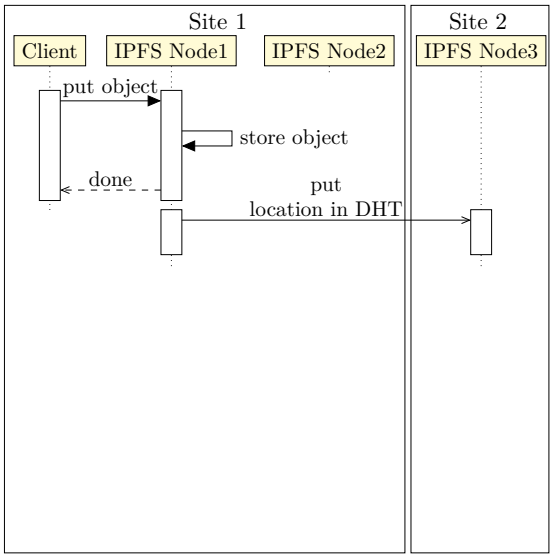
5.1.1 Description du problème

Bien que nous ayons déjà illustré quels sont les échanges réseaux effectués par IPFS dans la Section 4.1.3, nous détaillons les processus d'écriture et de lecture sur la Figure 5.0.

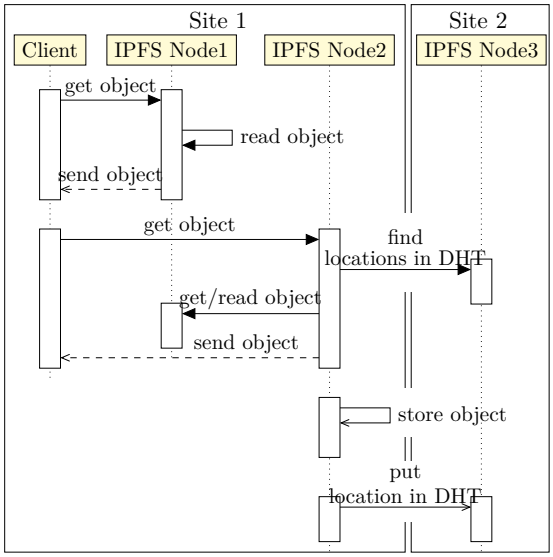
La Figure 5.1(a) montre la création d'un nouvel objet : le client envoie son objet à un nœud du site le plus proche. L'objet est créé localement et la table de hachage stockant la localisation de chaque objet est mise à jour. Comme la DHT ne fournit pas de localité, la localisation d'un objet peut être stockée sur n'importe quel nœud. Dans notre exemple le Nœud 3 appartenant au Site 2 stocke la localisation d'un objet qui a été écrit sur le Site 1.

La Figure 5.1(b) montre ce qu'il se passe lorsqu'un client lit un objet stocké sur le site local. Chaque fois qu'un nœud reçoit une requête pour un objet particulier, il vérifie d'abord si le nœud n'est pas stocké localement. Si cela n'est pas le cas, *(i)* un hash est calculé en fonction du nom de l'objet, *(ii)* le nœud stockant la localisation est contacté, *(iii)* l'objet est téléchargé en utilisant le protocole BitTorrent, *(iv)* une copie locale est créée pendant que l'objet est transmis au client et *(v)* la table de hachage distribuée est mise à jour pour refléter l'existence de ce nouveau réplica. Enfin, la Figure 5.0(c) décrit le protocole lorsque l'objet est demandé depuis un site distant (c'est-à-dire, lorsque le client s'est déplacé d'un site à l'autre). Dans tous les cas, le client contacte un nœud du site le plus proche.

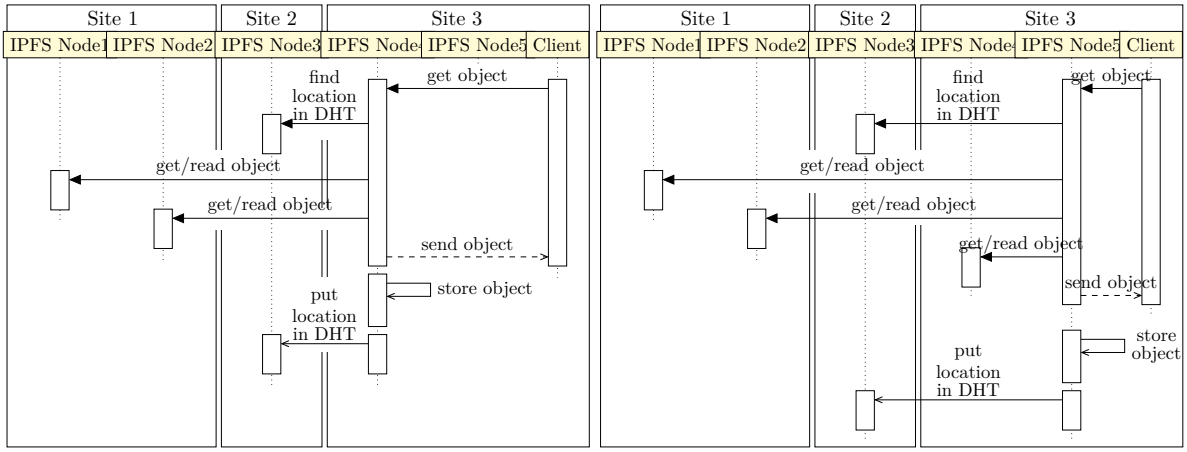
En d'autres mots, IPFS ne prend pas en compte la topologie physique de l'infrastructure de Fog et considère chaque serveur de façon indépendante plutôt que de considérer



(a) – Écriture d'un objet.



(b) – Lecture d'un objet stocké localement.



(c) – Lecture d'un objet stocké sur un autre site (en interrogeant IPFS Node4 ou IPFS Node 5).

FIGURE 5.0 – Diagrammes de séquence montrant les processus d'écriture (a) et de lecture d'un objet stocké sur le site local (b) ainsi que sur un site distant (c), avec la solution de stockage IPFS.

des groupes de serveurs placés sur chacun des sites. Quand un réplica est créé, que ce soit par un utilisateur ou par relocalisation, les autres nœuds IPFS du site ne sont pas informés de ce réplica et reposent sur la table de hachage distribuée pour le localiser. Cela est illustré par la seconde lecture de la Figure 5.0(c). Bien que l'objet est déjà disponible sur le Nœud 4 du Site 3, le Nœud 5 doit contacter la table de hachage distribuée pour le

localiser.

Ceci a deux principaux inconvénients. Premièrement les temps d'accès ainsi que la quantité de trafic échangé entre les sites sont augmentés. Deuxièmement, les clients ne peuvent accéder à des objets stockés sur le site lorsque celui-ci est déconnecté des autres (partitionnement du réseau) puisque la localisation de l'objet ne peut être trouvée. IPFS fournit donc une manière efficace d'accéder aux données grâce au protocole BitTorrent mais ne prend pas en compte les particularités d'une architecture de Fog Computing.

5.1.2 Solution proposée

Pour améliorer le fonctionnement d'IPFS dans un environnement de Fog Computing, proposons de déployer une solution de stockage de type *Scale-Out NAS* indépendamment sur chaque site. Ce *Scale-Out NAS* est utilisé comme solution de stockage sous-jacente à tous les nœuds IPFS d'un même site, comme cela est illustré sur la Figure 5.1. Cette approche permet à un nœud de stockage IPFS d'accéder aux objets stockés par les autres nœuds du même site, en ne faisant que de petites modifications dans le code source du programme. Ce couplage est possible car chaque nœud IPFS stocke de façon interne chaque objet sous la forme de fichiers. Au lieu d'accéder à un fichier stocké sur le disque dur local, IPFS ira simplement chercher l'objet demandé dans sur le système de fichiers distribué.

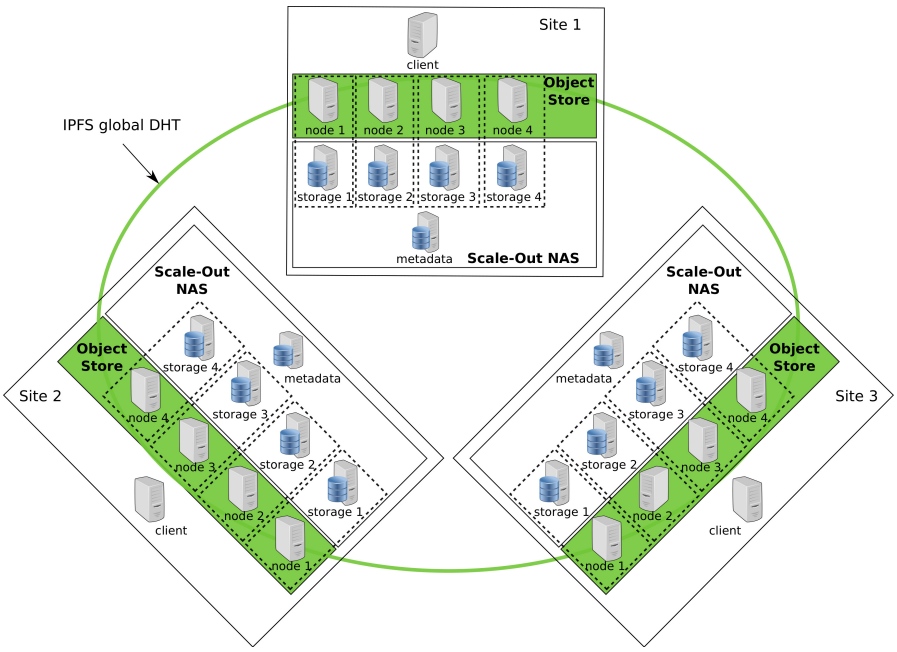


FIGURE 5.1 – Topologie utilisée pour déployer une solution de stockage en mode objet au-dessus d'un système de fichiers distribué localement sur chaque site.

Coupler un système de stockage objet avec un *Scale-Out NAS* n'est pas une idée nouvelle et de nombreuses propositions ont déjà été faites dans ce sens. Par exemple

Yahoo a développé Walnut [Chen et al. 2012], une solution de stockage par objets qui s'appuie sur différents systèmes de fichiers comme HDFS. Cependant, notre approche est différente dans le sens où dans notre topologie, le *Scale-Out NAS* sous-jacent n'est pas global mais local à chaque site. IPFS n'est qu'une « glue » pour créer un espace de noms global au travers des différents sites. Une approche similaire a été proposée dans le système Group Based FileSystem (GBFS) [Lebre et al. 2014] où différents systèmes de fichiers déployés à différents endroits sont agrégés. Cependant cette solution n'est pas efficace puisque tout accès commence par solliciter le serveur gérant la racine du système de fichiers, qui est alors fortement sollicité.

Enfin, l'intérêt d'utiliser un *Scale-Out NAS* comme RozoFS [Pertin et al. 2014] ou GlusterFS [Davies et al. 2013], est la possibilité de pouvoir ajouter des nœuds à la volée, ce qui augmente à la fois l'espace de stockage mais aussi les performances puisque dans un *Scale-Out NAS*, les requêtes et les données sont réparties de façon uniforme entre les nœuds de stockage. Les modifications du protocole utilisé par IPFS que nous proposons sont illustrées sur les Figure 5.1 et 5.2.

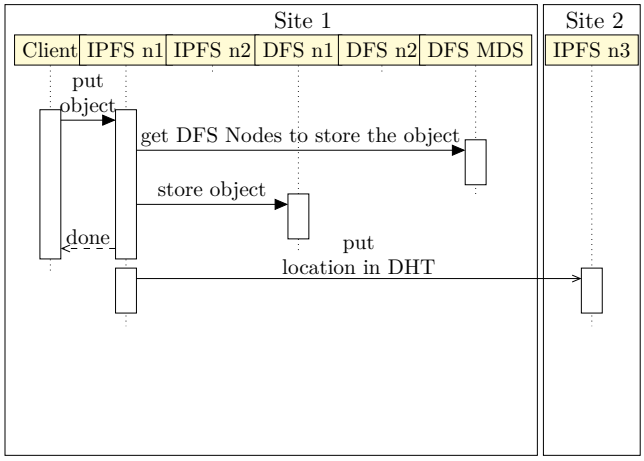
Nous pouvons observer qu'il n'y a aucun changement dans le protocole lors de la création d'un objet, si ce n'est qu'au lieu de stocker l'objet directement sur le nœud, celui-ci est envoyé dans un système de fichiers distribué.

Les principaux changements apparaissent lorsqu'un client veut accéder à une donnée stockée. La Figure 5.2(b), montre ces changements : grâce au *Scale-Out NAS*, tout nœud IPFS d'un site voit les objets manipulés par les autres nœuds du même site. En conséquence, peu importe à quel nœud du site les requêtes sont envoyées, lorsque le nœud interrogé recherchera si l'objet est stocké localement, il recherchera dans le *Scale-Out NAS* et l'enverra au client. Contrairement à la conception originale d'IPFS, il n'y a plus besoin d'accéder à la DHT pour localiser les objets stockés sur les autres nœuds d'un même site.

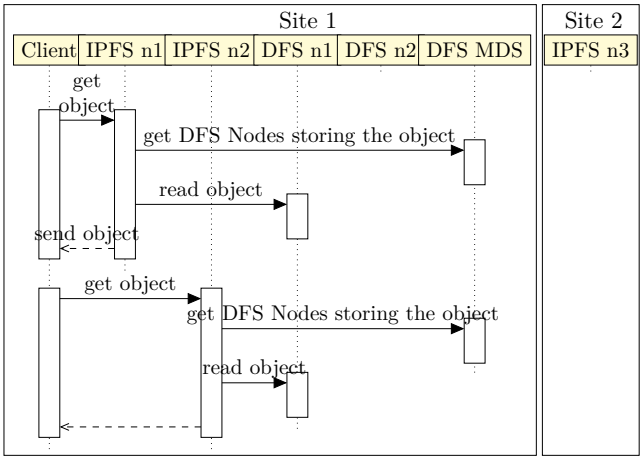
De manière similaire, la Figure 5.2(c) montre le protocole lors de l'accès à une donnée stockée sur un site distant. Lorsque le client est connecté sur un site qui ne stocke pas l'objet, la requête est reçue par n'importe quel nœud du site. Le nœud vérifie si l'objet n'est pas dans le système de fichiers distribué. Si l'objet n'est pas trouvé, la DHT est consultée pour localiser un réplica. L'objet est téléchargé, relocalisé dans le *Scale-Out NAS* et la table de hachage distribuée est mise à jour. Les futurs accès effectués depuis ce site, pourront être satisfaits sans aucune communication avec les sites extérieurs, peu importe le nœud du site qui sera interrogé. Nous devons enfin préciser que les objets utilisés par IPFS étant immuables, nous n'avons pas à nous préoccuper de savoir s'il est préférable de télécharger la version relocalisée dans le *Scale-Out NAS* local est plus récente que le réplica original stocké sur un site distant.

Pour résumer, notre approche permet de limiter la quantité de trafic réseau en cas de lecture d'objets stockés localement. Lorsqu'un client accède à un objet disponible sur le site, la table de hachage distribuée n'est pas utilisée, quel que soit le nœud du site interrogé. Dans les autres situations, la quantité de trafic réseau échangé reste la même.

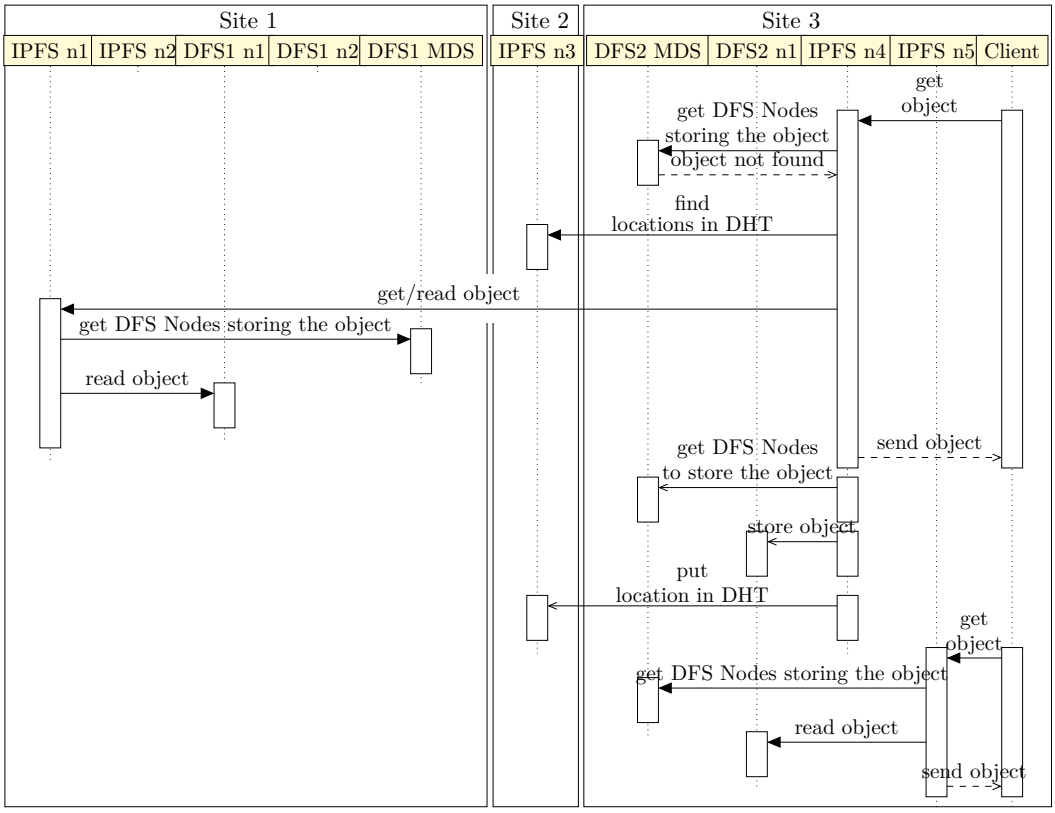
Un dernier point que nous devons préciser est que notre approche permet également de répartir les données stockées sur l'ensemble des nœuds de stockage du site de façon



(a) – Écriture d'un objet.



(b) – Lecture d'un objet stocké localement.



(c) – Lecture d'un objet stocké sur un site distant.

FIGURE 5.2 – Diagrammes de séquences montrant les processus de lecture et d'écriture d'un objet en utilisant IPFS au-dessus d'un système de fichiers distribué (DFS) déployé indépendamment sur chaque site. « MDS » (ou *MetaData Store*) est le serveur de métadonnées du système de fichiers distribué.

uniforme. En effet, si un client envoie une grande quantité de données à un seul nœud IPFS, dans l'approche standard, ce nœud devra stocker lui-même ces données, déséquilibrant la répartition des données au sein du site. Dans notre approche, les données sont réparties sur les différents serveurs de stockage du *Scale-Out NAS*, aboutissant à conserver à tout moment une répartition homogène sur les différents serveurs de stockage.

5.1.3 Limitations

D'un point de vue des performances, il est important de noter que l'approche que nous venons de présenter mérite d'être améliorée. En effet, dans l'approche par défaut, lorsqu'un nœud IPFS souhaite accéder à un objet situé sur un autre site, l'objet est téléchargé depuis l'ensemble des nœuds stockant un réplica (voir la Figure 5.1(b) de la Section 5.1.1). Ce comportement permet à IPFS de réaliser le téléchargement de la manière la plus efficace possible : si un nœud est surchargé, tous les autres nœuds sont capables de répondre.

Dans notre approche utilisant un *Scale-Out NAS*, la situation est différente car la DHT ne connaît qu'un nœud par site stockant le réplica : soit le nœud sur lequel le client a écrit l'objet, soit le premier nœud de chaque site à y accéder. Comme cela est illustré dans la Figure 5.2(c) de la Section 5.1.2, , le nœud IPFS 4 du Site 3 ne peut télécharger l'objet que du Nœud 1 du Site 1 car c'est sur ce nœud que l'objet a été écrit et c'est ce nœud qui est enregistré dans la DHT. Le nœud IPFS 4 du Site 3 ne sait pas qu'il peut aussi contacter le nœud IPFS 2 du Site 1. Cette situation peut mener à des goulets d'étranglements ou à la surcharge de certains nœuds. De la même manière si le nœud IPFS 1 du Site 1 devient inaccessible, l'objet ne peut plus être téléchargé du Site 1 ce qui est « gênant » dans la mesure où celui-ci est toujours accessible au travers des autres nœuds du site. Une solution pour ce problème serait de stocker dans la DHT non pas l'adresse du nœud mais un identifiant correspondant au site stockant l'objet. Nous laissons ces améliorations pour une contribution future.

5.1.4 Conclusions

Nous avons vu dans cette partie que l'utilisation d'une table de hachage distribuée pour localiser les objets ne permettait pas de garantir un confinement du trafic réseau : un site distant doit être contacté lorsque nous souhaitons accéder à un objet stocké sur le site local. Nous avons proposé de coupler IPFS avec un *Scale-Out NAS* local à chaque site permettait de donner une vue à chaque nœud de l'ensemble des objets stockés sur le site. De cette façon les objets stockés localement n'ont pas besoin d'être localisés en utilisant la table de hachage distribuée, réduisant les trafics réseau inter-sites et améliorant la disponibilité des données en cas de partitionnement du réseau.

Toutefois, avec l'utilisation d'une table de hachage distribuée, devoir contacter un premier site qui n'est pas nécessairement proche de l'objet accédé est un problème plus général qui ne se limite pas aux accès locaux. Dans la prochaine section, nous proposons de remplacer la table de hachage distribuée par une approche permettant de favoriser l'interrogation des sites voisins au sens de la topologie physique du réseau.

5.2 Confinement des trafics réseau inter-sites lors d'accès distants

Nous venons de voir que coupler IPFS avec un *Scale-Out NAS* déployé localement sur chaque site permettait de confiner le trafic lors d'accès locaux. Cependant pour les accès distants, l'utilisation d'une table de hachage distribuée ne permet toujours pas de confiner le trafic au plus près des utilisateurs et ne permet pas de stocker la localisation des données au plus proche de ces dernières. Nous proposons dans cette section de présenter brièvement différents mécanismes permettant de stocker la localisation de données, avant de proposer notre protocole de gestion des localisations.

Ce travail a été soumis dans la conférence IEEE Globecom 2018 [Confais et al. 2018a] et a été présenté à RESCOM 2018 [Confais et al. 2018c].

5.2.1 Approches pour localiser des données

Nous présentons ici, plusieurs méthodes permettant de localiser des données. Toutes se caractérisent sur la quantité de trafic échangé entre les sites lors du processus de localisation, par la garantie ou non de trouver un objet et par la localité des enregistrements, c'est-à-dire, la possibilité ou non de pouvoir placer l'enregistrement associant un nom d'objet à l'emplacement de ses réplicas proche des réplicas de l'objet.

Approches sans annuaire

Dans cette première section, nous présentons les méthodes pour lesquelles il n'y a pas besoin de créer un annuaire contenant la localisation de chaque donnée.

La première approche consiste à inonder le réseau pour trouver la donnée recherchée. Contrairement à l'approche utilisée par Cassandra où l'inondation est réalisée de façon asynchrone et dont le but est d'informer l'ensemble des nœuds de la topologie du réseau, ici, ce sont les requêtes des utilisateurs qui sont propagées dans le réseau.

La requête est transmise à l'ensemble du voisinage jusqu'à ce que le nœud stockant la donnée ou sa localisation soit atteint. Pour limiter les boucles, les requêtes sont assorties d'un TTL (*Time To Live*) permettant de limiter le nombre de sauts. Le principal défaut de cette méthode est qu'elle ne fournit pas de garantie pour trouver une donnée. Pire, un même client peut trouver une donnée lorsqu'il se trouve à une certaine position du réseau et ne plus y arriver après s'être déplacé. La Figure 5.3 illustre le cas où le client, situé au milieu du réseau effectue une recherche par inondation en limitant le nombre de sauts (TTL) à 3. Ces voisins émettent à leur tour la requête à leurs voisins (sauf vers le nœud d'où ils ont reçu la requête) avec un TTL égal à 2. Et enfin ces derniers nœuds réémettent la requête avec un TTL égal à 1. Le nombre de sauts n'étant pas suffisant pour atteindre le nœud stockant la donnée, le client restera à penser que la donnée recherchée est inexistante dans le réseau. Augmenter le nombre de sauts permettrait de trouver la donnée mais effectuer un nombre de sauts élevé non seulement augmente les temps d'accès mais aussi génère un trafic conséquent sur le réseau. Dans certains cas où l'utilisateur reste proche du nœud stockant les données accédées, cette approche peut

être utilisée [Shah et al. 2005a]. Toutefois, transférer les requêtes de proche en proche, en inondant le réseau ne permet pas de respecter notre critère du confinement du trafic réseau présenté dans la Section 3.1. Certaines heuristiques permettent de limiter le trafic réseau tout en essayant de maximiser les chances de trouver la donnée recherchée [Rahman et al. 2004 ; Gkantsidis et al. 2004 ; Shah et al. 2005b].

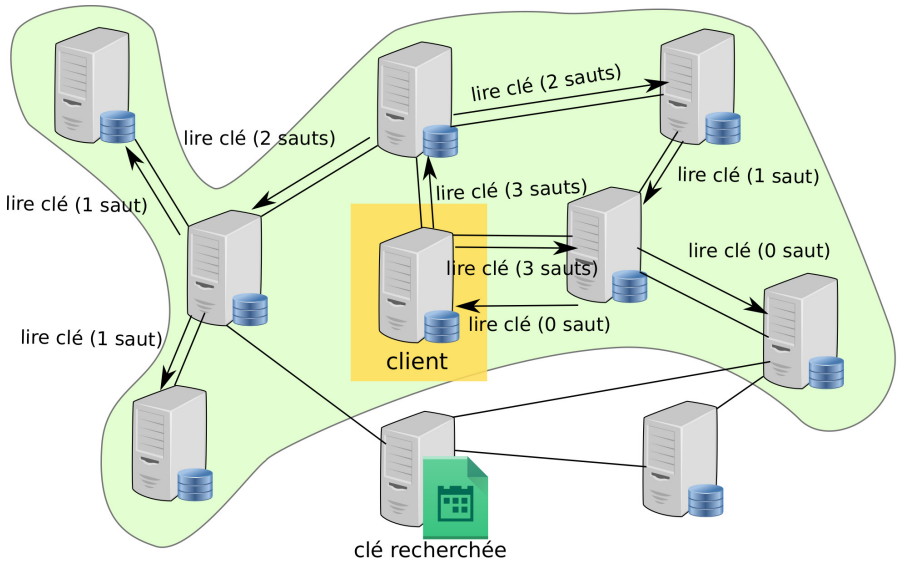


Figure 5.3 – Protocole par inondation qui ne garantit pas de pouvoir accéder à la donnée recherchée. Le nuage vert contient les nœuds qui ont été atteints par la requête.

De façon similaire, certains protocoles de routage réseau tel qu'*Optimized Link State Routing Protocol* (ou *OLSR*) [Clausen et al. 2003] utilisent également une approche par inondation préalablement à l'échange de table de routage.

Une autre approche est de stocker la localisation de la donnée dans son identifiant [Gifford et al. 1988 ; Satyanarayanan 1990]. Le système de stockage n'a pas besoin de retenir la localisation puisque l'utilisateur l'indique à chaque fois qu'il souhaite accéder à une donnée. La contrepartie de faire reposer le processus de localisation sur les utilisateurs eux-mêmes est que lorsque la donnée est déplacée, celle-ci change de nom. Un tel système semble compliqué à utiliser à l'usage. Il est en revanche utile lorsque les données ne sont jamais déplacées. Une telle approche serait similaire à ce qu'il se passe aujourd'hui dans le routage réseau, où l'adresse IP identifie de manière unique une machine mais dépend du réseau auquel cette machine est connectée. Lorsque la machine se déplace, son adresse change. Comme mentionné dans la Section 2.4.4, des protocoles comme LISP, bien qu'ayant des difficultés de passage à l'échelle, visent à séparer ces deux utilisations et permettre d'avoir un identifiant constant qui ne dépend pas de la position du nœud dans le réseau.

Fonctions de hachage

Les fonctions de hachage peuvent être utilisées pour déterminer le nœud stockant la localisation d'une donnée. Une fonction de hachage est appliquée à l'identifiant de la donnée à stocker. La fonction retourne non pas l'identifiant du nœud stockant l'objet mais l'identifiant du nœud stockant la localisation de l'objet. Cela est illustré par la Figure 5.4. Le client exécute une fonction de placement afin non pas de localiser la donnée, mais de localiser le nœud stockant la localisation de la donnée.

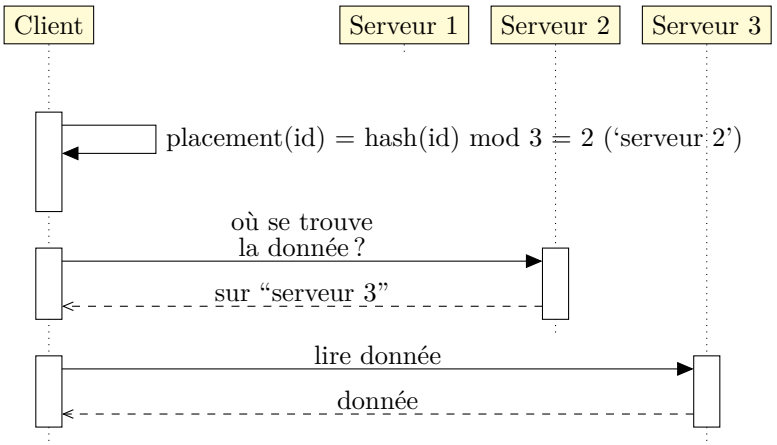


FIGURE 5.4 – Diagramme de séquence montrant comment une fonction de hachage peut être utilisée dans le processus de localisation d'une donnée.

L'inconvénient d'une telle approche est le même que lorsque la fonction de hachage est directement utilisée pour placer les données : en cas de changement de la topologie, certains enregistrements de localisation doivent être déplacés mais surtout, tous les clients doivent également être prévenus de cette nouvelle topologie pour être capable de trouver la nouvelle localisation des identifiants des objets.

Il y a également un problème de localisation : lorsque le client calcule $hash(id)$ qui retourne 2, le client a besoin de savoir qui est le serveur 2 et comment l'atteindre. En d'autres mots, le client a besoin d'un mécanisme qui associe une adresse IP au serveur 2 ou d'une méthode de routage comme cela sera proposé dans les tables de hachage distribuées que nous avons présentées dans la Section 4.1.2.

Lorsque le nombre de serveurs de stockage change, l'emplacement de chaque objet ou chaque localisation doit être recalculé. Les fonctions de hachage consistantes, comme CRUSH utilisée dans Rados [WEIL et al. 2007], permettent de limiter la quantité de données déplacées lors d'un changement de topologie. Afin que chaque objet puisse être localisé à tout moment par chaque serveur du réseau, il est nécessaire de diffuser la topologie du réseau de manière consistante à l'ensemble des nœuds. Pour cela, des systèmes de stockage comme Tahoe-LAFS [SELIMI et al. 2014] utilisent un serveur centralisé quand Rados recourt à l'algorithme Paxos [LAMPORT 2002], qui est un algorithme permettant de réaliser un consensus distribué. De cette façon, tous les nœuds travaillent au même

moment avec la même version de la topologie réseau. Le problème est que Paxos est un algorithme qui passe difficilement à l'échelle, ce qui rend son utilisation compromettante pour une utilisation dans un contexte de Fog Computing comme nous avons pu le mettre en évidence sur le Chapitre 4.

5.2.2 Tables de hachages distribuées

Les tables de hachage proposent à la fois de déterminer le nœud stockant la localisation de la donnée mais également en proposant un mécanisme de routage vers ce nœud (grâce à un réseau de recouvrement). Un espace de clés est organisé en anneau et à chaque nœud est attribué, grâce à un tirage aléatoire, une position sur l'anneau. Les nœuds stockent les clés dont le *hash* est compris entre leur position et la position de leur voisin. De cette façon, en faisant circuler les requêtes autour de l'anneau, celles-ci finiront par arriver sur les nœuds stockant la localisation des objets concernés. Toutefois, afin d'accélérer le routage, les tables de hachage distribuées proposent de construire des tables de routage au sein de chaque nœud : au lieu de transmettre la requête à leur voisin, les nœuds peuvent la transmettre au nœud qu'ils estiment être le plus proche du nœud de destination. La plupart des tables de hachage distribuées garantissent un nombre logarithmique de sauts (par rapport au nombre de nœuds) avant d'atteindre la destination finale.

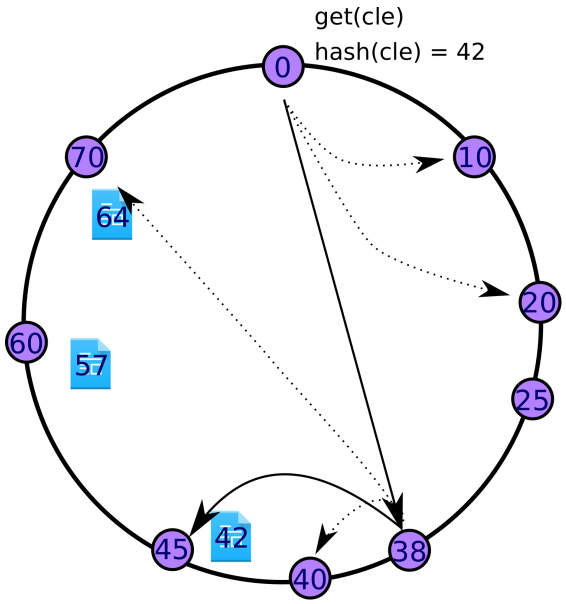


FIGURE 5.5 – Exemple de table de hachage distribuée de type Chord. Les ronds violets représentent les nœuds avec leur identifiant. Le nœud d'identifiant 0 effectue une requête pour accéder à la clé `cle`, d'identifiant 42.

De nombreuses tables de hachage distribuées ont été proposées au fil des ans. Par exemple, CAN [RATNASAMY et al. 2001], Kademlia [MAYMOUNKOV et al. 2002] Pastry [ROWSTRON et al. 2001], Chord [STOICA et al. 2003] ou Tapestry [ZHAO et al. 2006]

et de nombreux systèmes de fichiers s'y appuient [Dabek et al. 2001 ; Muthitacharoen et al. 2002 ; Ashraf et al. 2012]. Le principal avantage de ce réseau de recouvrement est de proposer un réseau dynamique, permettant de gérer automatiquement l'ajout et la suppression des nœuds (ou *churn*) [Rhea et al. 2004 ; Hildrum et al. 2004]. Toutefois, router les requêtes, même avec un nombre logarithmique de sauts, nécessite du temps et réduit les performances.

La Figure 5.5 montre un exemple de routage au sein d'une table de hachage distribuée ou *Distributed Hash Table* (DHT) de type Chord. Deux sauts sont nécessaires au nœud d'identifiant 0 pour accéder à la clé `cle`. Les flèches correspondent aux routes contenues dans la table de hachage du nœud d'identifiant 0 et du nœud intermédiaire d'identifiant 38.

Cette approche multi-sauts peut augmenter les temps d'accès, c'est pourquoi les tables de hachage à un seul saut ont été proposées. Ces tables de hachage à un saut, comme celle utilisée par Cassandra, font en sorte que chaque nœud connaisse la portion de l'espace de clé dont chaque nœud est responsable. De cette façon, chaque requête peut être transmise directement au nœud de destination.

Afin de distribuer cette information à l'ensemble des nœuds, Cassandra repose sur une approche par inondation (ou *gossip*). À intervalles réguliers, chaque nœud envoie à un nœud sélectionné aléatoirement, sa connaissance de la portion de l'espace de clé géré par chacun des nœuds.

Le principal inconvénient des tables de hachage est leur manque de localité : le nœud stockant la localisation d'une donnée ne peut pas être choisi manuellement. Il est le résultat d'une fonction de hachage. Par conséquent, dans un contexte de Fog Computing, il faut accéder à un premier site pour localiser la donnée que nous voulons accéder, puis une fois la donnée localisée, accéder au site sur lequel celle-ci est stockée. Dans un contexte de Fog Computing, cet accès à un site supplémentaire ne respecte pas nos contraintes de « localité » et de « confinement du trafic réseau ». L'avantage des tables de hachage distribuées par rapport aux approches par gossip est qu'il n'y a pas besoin d'attendre que tous les sites aient répliqués la localisation pour pouvoir y accéder. En revanche, tout comme l'approche par hachage, ne pas choisir le nœud stockant une clé donnée est un sérieux inconvénient.

Nous nous intéressons à introduire de la localité dans les tables de hachage distribuées. Le but de cette section est de rechercher s'il est possible de contrôler le nœud devant stocker la localisation d'une donnée ou bien s'il existe des stratégies de réplication permettant de répliquer cette localisation près des nœuds qui émettent les demandes.

Plusieurs approches ont été proposées pour améliorer l'efficacité et les performances des tables de hachage distribuées. Cela passe notamment par l'augmentation du facteur de réplication mais aussi par une exécution récursive des requêtes : lorsqu'un nœud n'est pas la destination de la requête, il transmet lui-même la requête au nœud qui lui semble le plus approprié au lieu de demander au nœud initial d'interroger lui-même ce nœud [Dabek et al. 2004a][Kang et al. 2009].

Un premier aspect de la localité dans les tables de hachage distribuées est la localité du routage. Cela correspond à l'idée que lorsqu'un nœud d'identifiant x accède à une clé

stockée sur un nœud d'identifiant y, la requête ne peut transiter que par les nœuds dont l'identifiant est compris entre x et y [Zhao et al. 2002]. De cette façon, plus un nœud est « proche », moins il faut de sauts pour l'atteindre. Cette idée est utilisée dans les tables de hachage utilisant un routage de type Plaxton [Plaxton et al. 1996 ; Plaxton et al. 1999], comme Pastry [Rowstron et al. 2001], Tapestry [Zhao et al. 2006], ou Kademlia [Maymounkov et al. 2002]. Pour réaliser cela, l'approche utilisée consiste à effectuer un routage selon des suffixes : chaque nœud transmet la requête au voisin ayant l'identifiant maximisant le suffixe commun avec l'identifiant de la clé recherchée. Dans la Figure 5.6, le nœud d'identifiant 9098 route la requête à destination du nœud d'identifiant 4**598** vers le nœud **7598** car l'identifiant de ce dernier possède les trois derniers chiffres en commun avec l'identifiant du nœud de destination.

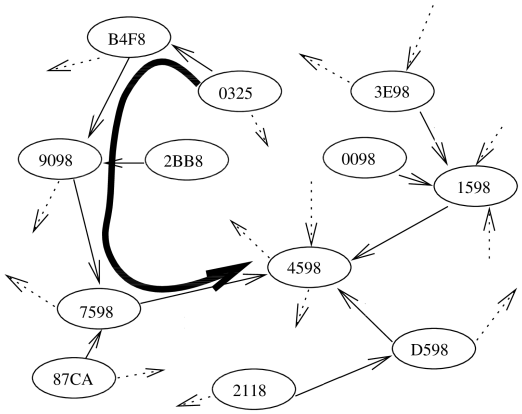


Figure 5.6 – Exemple de routage par préfixe. Figure extraite de : Tapestry : An Infrastructure for Fault-tolerant Wide-area Location and Routing [Zhao et al. 2006]

Le problème d'une telle approche est qu'elle ne tient pas compte de la topologie réseau sous-jacente : deux nœuds ayant un identifiant proche dans la table de hachage distribuée peuvent ne pas être physiquement proches au niveau réseau. Cela est illustré par la Figure 5.7 illustrant une table de hachage distribuée, construite sous forme d'anneau et déployée dans le réseau RENATER (Réseau National de télécommunications pour la Technologie l'Enseignement et la Recherche).

Pour tenir compte de ce phénomène, deux types d'approches sont proposées. Kademlia propose, en plus de router vers le nœud ayant le plus grand préfixe commun, de router vers le nœud ayant la plus faible latence [Kaune et al. 2008]. Dans l'exemple précédent, cela revient pour le nœud 9098 à préférer transmettre la requête vers le nœud 7598 plutôt que le nœud d'identifiant 9598 si ce dernier ne peut être contacté qu'avec une latence plus élevée (les nœuds 7598 et 9598 ont un suffixe de taille identique pour la requête à destination du nœud 4598). Cette approche est reprise dans plusieurs tables de hachage distribuées [Wu et al. 2008].

Une autre approche consiste à utiliser des nœuds virtuels. Tous les nœuds proches

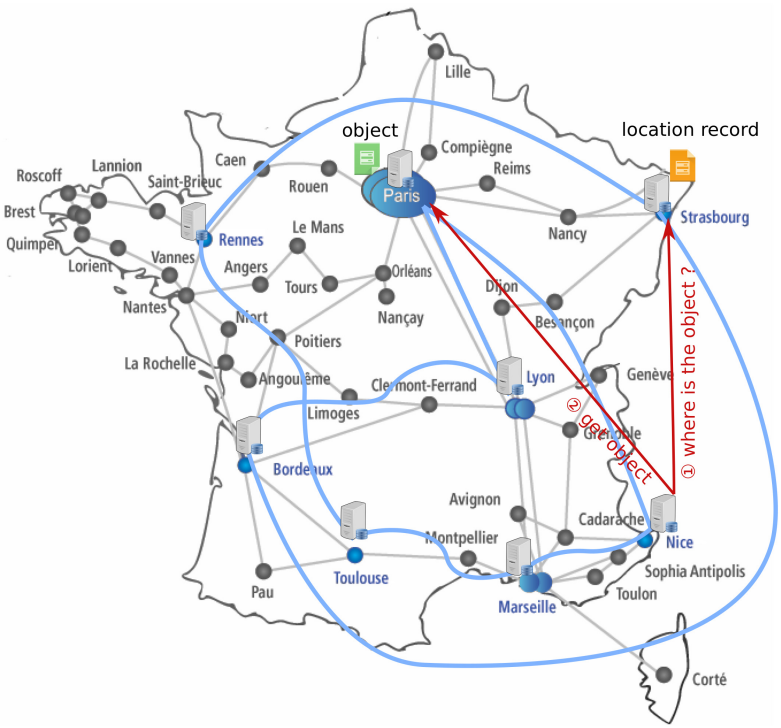


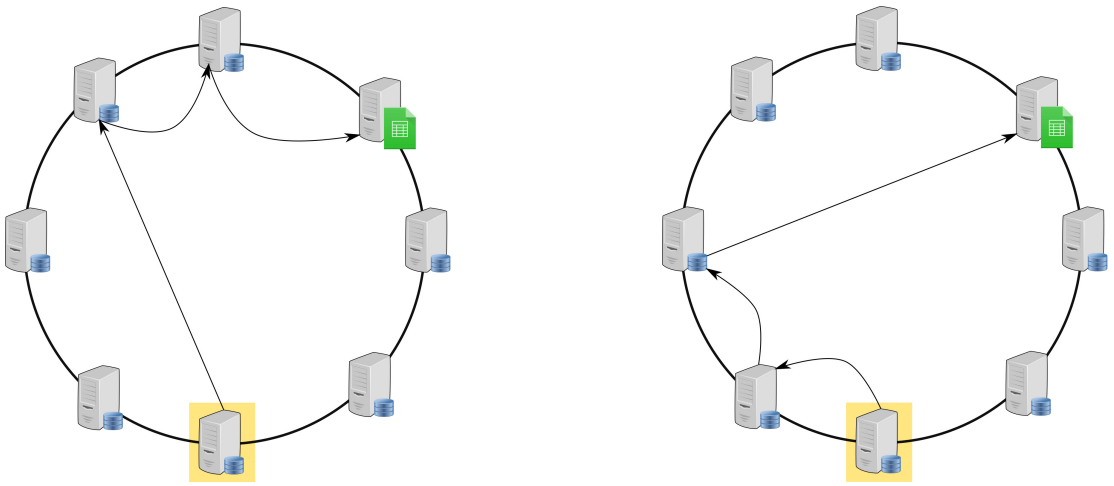
FIGURE 5.7 – Exemple d'un anneau DHT (en bleu) et échanges observés lorsqu'un objet stocké à Paris est accédé depuis Nice (en rouge). La topologie réseau utilisée est celle du réseau RENATER.

géographiquement sont regroupés au sein d'un seul nœud de façon à faire face à la dynamique du réseau mais aussi de pouvoir contacter directement un nœud physiquement proche qui stockerait la clé demandée [ZAHN et al. 2005] ou bien d'essayer de découvrir la topologie physique du réseau, en appuyant la construction de l'anneau sur un protocole comme Vivaldi [DABEK et al. 2004b] ou bien directement sur un protocole de routage dynamique [FANTAR et al. 2009].

Enfin, d'autres propositions visent à choisir le routage dans la table de hachage [CASTRO et al. 2003 ; SERBU et al. 2011]. Par exemple, se rapprocher le plus du nœud de destination lors des premiers « sauts » ou des derniers, ou encore prendre en compte la latence réseau lors de la sélection du nœud sur lequel le « saut » est fait. Cela est illustré par la Figure 5.8.

La localité des réplicas consiste à placer les réplicas de la clé, « proches » des nœuds qui y accèderont plutôt que d'optimiser l'accès à un nœud potentiellement situé à l'autre bout du réseau. Dit autrement, à partir de la Figure 5.7, Nice devrait localiser l'objet en contactant l'un des sites, physiquement proche de lui plutôt que de contacter un nœud situé à Strasbourg, et ce, quelle que soit la façon dont l'anneau de la table de hachage distribuée est construit.

Traditionnellement, les clés sont répliquées sur les nœuds qui, dans l'anneau, suivent



(a) – Stratégie de routage effectuant les plus « grands sauts » en premier.

(b) – Stratégie de routage effectuant les plus « petits sauts » en premier.

FIGURE 5.8 – Deux exemples de stratégies de routage dans une table de hachage distribuée.

ou précédent le nœud responsable de celles-ci [STOICA et al. 2003]. Cependant, certaines stratégies de réplication peuvent permettre de trouver un réplica plus « proche ». Une idée très souvent proposée est lorsqu'un nœud accède à une clé, il dépose un réplica sur chacun des nœuds contactés pour atteindre cette clé. De cette façon, les requêtes futures pourront être résolues par les nœuds intermédiaires, sans avoir besoin d'atteindre la destination finale [KTARI et al. 2007 ; KIM et al. 2007 ; CHANDY 2008].

Choisir l'identifiant des clés à stocker est une solution très rarement proposée. Dans une table de hachage utilisant un routage de type Plaxton, suffixer l'identifiant de la donnée par un suffixe partagé par des nœuds proches permet de garantir que la localisation de la donnée sera stockée sur l'un de ces nœuds. Le prix à payer est que la localisation se retrouve dans le nom de l'objet. Cette approche est notamment explorée par Harvey *et al.* [HARVEY et al. 2003]. Nous noterons que dans une telle approche, la fonction de « hachage » n'est plus présente dans la DHT, le placement est directement fait sur la clé et non sur un hash de celle-ci.

Enfin, des approches hiérarchiques ont été proposées [GARCÉS-ERICE et al. 2003]. L'idée est de séparer les nœuds en plusieurs groupes. Chaque groupe contient une table de hachage distribuée locale au groupe et un super-nœud. Le super-nœud annonce dans une table de hachage globale quelles sont les clés stockées au sein du groupe. Cela permet aux nœuds d'un même groupe d'accéder aux clés stockées au sein du groupe sans interroger les nœuds situés en dehors. Cela permet également de s'assurer qu'un nœud d'un groupe stocke ses clés au sein de ce même groupe. Bien que les tables de hachage locales à chaque groupe permettent d'assurer un peu de localité, la table de hachage globale continue de ne fournir aucune localité. De nombreuses solutions combinent différentes approches

présentées précédemment. Par exemple FlowerCDN [El Dick et al. 2009] utilise une table de hachage distribuée entre les sites et un serveur de métadonnées centralisé au sein de chaque site. La table de hachage distribuée fait l'association entre un dossier et un site puis dans un second temps, un serveur de métadonnées, local à chaque site, permet de déterminer le serveur stockant le dossier. D'autres auteurs remplacent le serveur centralisé local à chaque site par un protocole de gossip [Vasilakos et al. 2015]. Zaharia *et al.* [Zaharia et al. 2008] proposent de rechercher une donnée en parcourant le réseau par voisinage. Si la donnée n'est pas trouvée, une table de hachage distribuée prend le relai.

Pour résumer, aucune proposition ne fournit à la fois :

- Un réseau de recouvrement qui respecte la topologie physique ;
- Un routage « optimal », limitant le nombre de nœuds à contacter lors de l'exécution d'une requête (localité du routage) ;
- Ne pas reposer sur une fonction de hachage pour pouvoir choisir le nœud sur lequel une clé est stockée.

Approches par gossip

L'idée d'une approche par gossip est que le nœud voulant accéder à une donnée connaisse directement la localisation de cette dernière. Les nœuds s'échangent régulièrement la liste des objets qu'ils stockent de façon que chaque nœud puisse construire son propre annuaire.

Dans Dynamo [DeCandia et al. 2007], les nœuds échangent régulièrement la liste des données qu'ils stockent avec leurs voisins. Au bout de plusieurs échanges, l'ensemble des nœuds connaît l'emplacement de l'ensemble des données et peut donc directement y accéder. L'inconvénient d'une telle approche est que chaque nœud doit retenir l'ensemble des localisations, comme cela est illustré sur la Figure 5.9.

Pour palier cela, les approches par gossip sont parfois intégrées dans des « tables de hachage à un saut » [DeCandia et al. 2007 ; Li et al. 2013], comme cela est fait dans Cassandra [Lakshman et al. 2010]. Les données sont regroupées dans des espaces de clés, qui sont des ensembles de données stockées sur les mêmes nœuds. Cela permet de diffuser et de ne retenir uniquement la localisation de ces groupes, plutôt que de chaque objet individuel. Lorsqu'un utilisateur accède à une donnée, il doit alors spécifier en plus de l'identifiant de la donnée, le nom du groupe dans lequel celle-ci est stockée.

Dans le cas du Fog, cela signifie que les sites s'échangent régulièrement des informations. Mêmes les sites qui ne sont pas sollicités seraient impactés, ce qui ne respecte pas le critère de confinement de trafic réseau énoncé précédemment dans la Section 3.1. De plus une approche similaire à Cassandra reviendrait à avoir autant de groupes que de sites et *in fine*, à spécifier le nom du site sur lequel la donnée est stockée.

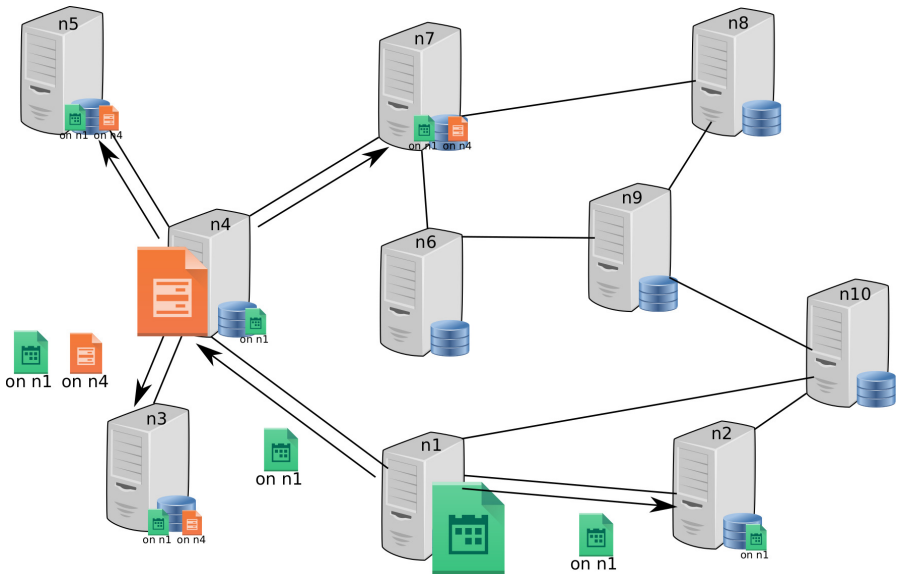


FIGURE 5.9 – Protocole par *gossip*. Chaque nœud transmet à son voisinage la liste des objets qu'il stocke ainsi que sa connaissance sur la localisation des objets stockés par les autres nœuds. Après un temps de convergence, l'ensemble des nœuds du réseau connaît l'emplacement de chaque objet et peut donc y accéder directement.

Chaînes de blocs

Les chaînes de blocs sont une autre proposition pour stocker la localisation des données [WILKINSON et al. 2014 ; VORICK et al. 2014]. C'est une structure de données qui ne fait que grossir. Les nœuds essaient de résoudre un problème mathématique dont les réponses ne peuvent se trouver que par force brute (preuve de travail) comme par exemple obtenir un certain nombre de 0 en préfixe d'un hash issu d'une fonction cryptographique (par exemple SHA-256, MD5). Chaque nœud qui trouve une solution au problème construit un bloc dans lequel il peut écrire des informations de toute sorte. Par exemple, des transactions monétaires comme dans l'application BitCoin [NAKAMOTO 2008 ; ZOHAR 2015] ou encore des programmes informatiques comme dans Ethereum [WOOD et al. 2013 ; DANNEN 2017]. Dans le cas d'une solution de stockage, l'information écrite dans le bloc peut correspondre à la localisation d'une donnée stockée. Le bloc est ensuite échangé par voisinage : chaque nœud transmet le bloc à ses voisins. Lorsqu'un nœud reçoit le bloc, il vérifie que la solution au problème est valide et n'avait pas encore été trouvée, puis, si tel est le cas, le bloc est ensuite ajouté dans la chaîne. Comme l'approche précédente, tous les nœuds ont la connaissance de la localisation de tous les objets. De plus, la chaîne de blocs est une structure dans laquelle il n'est pas possible de supprimer ou modifier les données. Lorsqu'une donnée est déplacée, la chaîne conservera l'historique de toutes les positions précédentes. Au-delà du problème de stockage de la localisation de tous les objets, le principal défaut de la chaîne de blocs est la vitesse à laquelle les données

peuvent être insérées. Les solutions au problème mathématique est un facteur limitant ce qui fait que les chaînes de blocs sont souvent limitées en moyenne à 7 transactions simultanées [Croman et al. 2016 ; Karame 2016]. Il faut parfois plusieurs minutes pour valider une transaction. Les chaînes de blocs utilisant une preuve de stockage ou de participation plutôt que l'utilisation d'une preuve de travail ont été proposées [Croman et al. 2016].

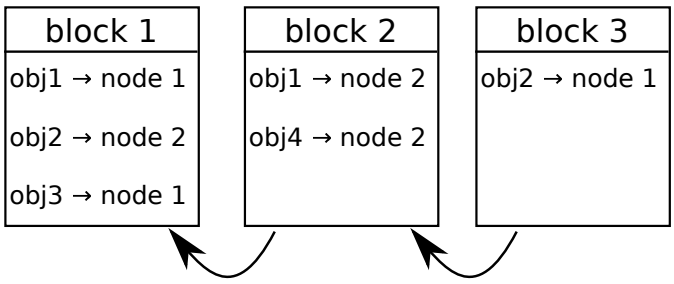
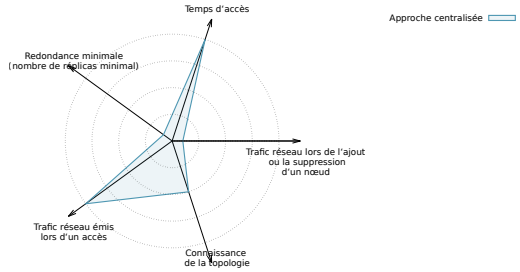


Figure 5.10 – Exemple de chaîne de bloc stockant les localisations des objets au fur et à mesure de leur déplacement. L'objet 2 a été déplacé du nœud 2 au nœud 1 et l'objet 1 du nœud 1 au nœud 2.

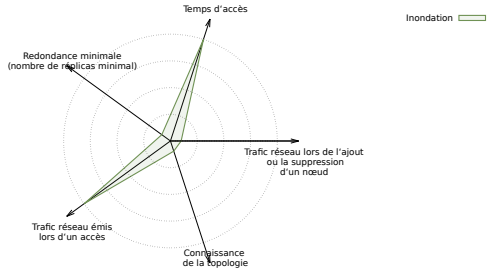
La Figure 5.10 montre un exemple d'une chaîne de blocs, devant sauvegarder toutes les anciennes positions d'une donnée. Par exemple, le bloc 1 stocke l'enregistrement indiquant que l'objet 1 est stocké sur le nœud 1. Lorsque l'objet est déplacé sur le nœud 2, un nouvel enregistrement est créé et est inséré dans le bloc 2 mais l'enregistrement se trouvant dans le bloc 1 est toujours présent. Ainsi, dans l'exemple présenté, 6 enregistrements sont stockés alors qu'il n'y a que 4 objets écrits. Dans un contexte de Fog, une telle approche qui réplique la localisation de chaque objet sur chaque nœud est coûteux et ne permet pas de « confiner le trafic réseau ». Certains travaux proposent d'utiliser des chaînes de blocs pour mettre en commun les ressources fournies par les objets de l'Internet des Objets [Samaniego et al. 2016] tandis que d'autres pensent à coupler un système de stockage tel qu'IPFS à une chaîne de blocs [García-Barriocanal et al. 2017] dans le but de ne pas avoir à stocker dans la chaîne de blocs des enregistrements de localisation qui ne sont plus d'actualité.

Conclusion

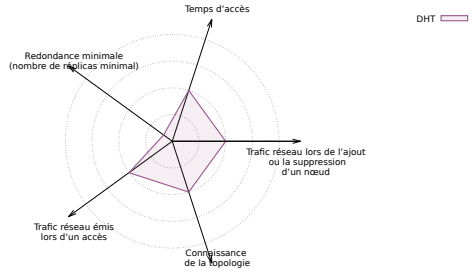
Aucune de ces approches ne permet de confiner le trafic réseau tout en contrôlant l'emplacement de la localisation. Soit la localisation est stockée sur un nœud sélectionné par hachage, soit elle est répliquée sur l'ensemble des nœuds (hormis dans le cas des tables de hachage distribuées). La Figure 5.11 synthétise les caractéristiques de chaque approche permettant de stocker la localisation des données. Plus une valeur est élevée, plus cela signifie que les temps d'accès sont élevés, que du trafic réseau est généré, que chaque nœud doit connaître une partie importante de la topologie ou qu'un enregistrement doit être répliqué sur de nombreux nœuds pour pouvoir être utilisé. Dit autrement, plus une



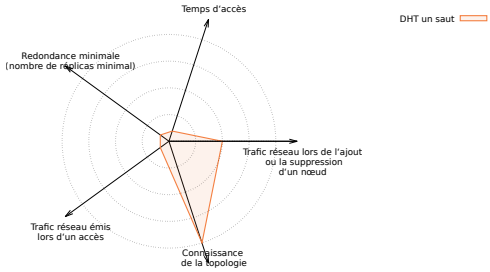
(a) – Approche centralisée



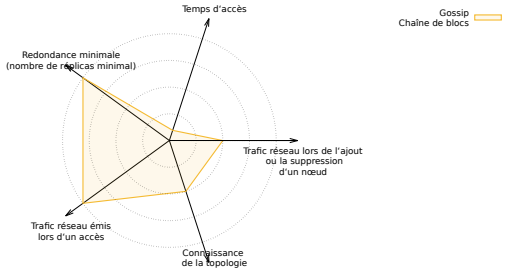
(b) – Inondation



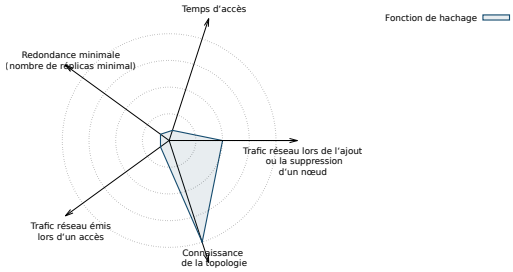
(c) – Table de hachage distribuée



(d) – Table de hachage distribuée à un saut



(e) – Gossip / Chaîne de blocs



(f) – Fonction de hachage

FIGURE 5.11 – Diagramme en étoile résumant les caractéristiques de plusieurs mécanismes permettant de localiser des données. Plus une valeur est faible, plus la propriété est compatible avec une infrastructure de Fog Computing.

valeur est élevée, plus le système aura des difficultés à fonctionner dans un environnement de Fog Computing.

Le critère de la redondance minimale ou du nombre de réplicas minimal permet de savoir si l'information de localisation n'est placée que sur un certain nombre de nœuds ou si elle doit être répliquée sur l'ensemble des nœuds. La propriété de « nombre de réplicas minimal » indique si le protocole a une contrainte sur le nombre de réplicas de localisation. Par exemple dans une approche par « gossip », les enregistrements sont répliqués sur l'ensemble des nœuds, générant pour chaque objet, autant de réplicas de localisation que de nœuds dans le réseau.

La connaissance de la topologie indique le nombre de nœuds connus dans les tables de routage. Par exemple avec une table de hachage distribuée, chaque nœud connaît $log(N)$ autres nœuds (avec N le nombre total de nœuds), tandis que dans une approche utilisant une fonction de hachage, chaque nœud doit connaître l'ensemble des autres nœuds. Pour l'approche centralisée, nous avons considéré que le nœud central connaît la topologie dans son ensemble. Nous aurions pu considérer que chaque nœud de stockage ne connaît que l'adresse du serveur centralisé et ainsi indiquer que l'approche centralisée se comporte bien vis-à-vis de ce paramètre.

Nous pouvons retrouver les propriétés d'un système de stockage pour le Fog à partir des caractéristiques utilisées dans la Figure 5.11. Par exemple, un système de stockage qui émet beaucoup de données sur le réseau lors d'un accès, ne confinera pas le trafic. De même, un système dans lequel la localisation de chaque objet est répliqué sur chaque nœud pourra supporter le mode déconnecté. Nous proposons dans le Tableau 5.1 une correspondance entre les propriétés de fonctionnement des mécanismes de localisation et les caractéristiques que nous avons proposées pour une solution de stockage dans une infrastructure de Fog Computing. Par exemple, un mécanisme de localisation qui génère du trafic réseau lors de l'accès de chaque objet, ne pourra pas confiner le trafic entre les sites de Fog. Une approche dans laquelle, la localisation de chaque objet est répliquée sur tous les nœuds pourra permettre un fonctionnement en mode déconnecté. Enfin, lorsque l'ensemble des nœuds doit connaître la topologie du réseau, cela peut créer des difficultés de passage à l'échelle.

Caractéristique d'une solution de stockage pour le Fog	Propriété du mécanisme de localisation des données
Confinement du trafic réseau	Trafic réseau échangé lors d'un accès
Mode déconnecté	Nombre minimal de réplicas
Passage à l'échelle	Connaissance exhaustive de la topologie par chaque nœud

Table 5.1 – Tableau montrant les correspondances entre les propriétés de la Figure 5.11 et les caractéristiques pour une solution de stockage pour le Fog.

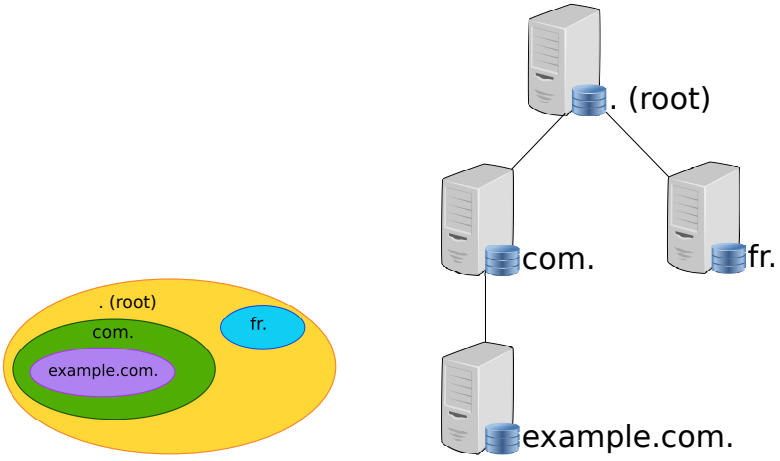
Le dernier mot de cette conclusion est qu'aucune de ces approches étudiées n'est adaptée pour le Fog. Dans la prochaine section, nous présenterons en quoi le protocole de nommage *Domain Name System* (ou DNS) peut nous permettre de confiner le trafic

lors d'accès à des objets stockés sur un site distant.

5.2.3 Le *Domain Name System* : un protocole de résolution de noms

Nous proposons de nous inspirer du protocole *Domain Name System* (DNS) utilisé sur l'Internet pour, entre autres, associer une adresse IP à un nom d'hôte. Ce protocole a l'avantage de passer à l'échelle et de permettre aux utilisateurs de « choisir » sur quel serveur doit être placé l'enregistrement voulu, en créant les délégations correspondantes.

Le Domain Name System (DNS) est une base de données distribuée [Mockapetris 1987]. L'espace de clés est hiérarchique : les serveurs sont organisés en arbre. Chaque serveur gère un ou plusieurs sous-espaces des clés. Pour chaque sous-espace, il peut stocker les clés de ce sous-espace ou bien le diviser à nouveau en d'autres sous-espaces et déléguer leur gestion à d'autres serveurs. Le caractère « . » est utilisé dans les clés comme caractère de séparation entre un espace et un sous-espace.



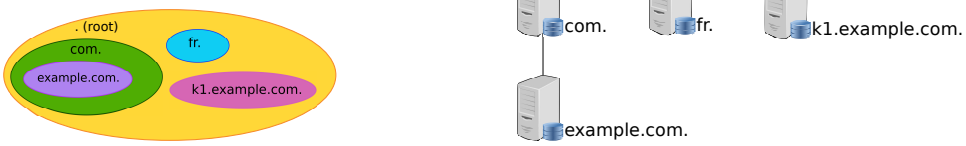
(a) – Découpage de l'espace de clés

(b) – Arbre

Figure 5.12 – Exemple de découpage de l'espace de clés et son affectation à l'arbre.

Le serveur racine s'occupe de la zone notée « . », représentant l'ensemble de l'espace de clés. Cet espace de clés est divisé en sous-espaces, délégués à plusieurs serveurs différents. Ainsi les clés du sous-espace « com. » n'est pas géré par le même serveur que les clés du sous-espace « fr. ». Chacun de ces serveurs redécoupe l'espace de clés et délègue la gestion de chaque domaine à un serveur différent. Un tel déploiement est illustré par la Figure 5.12. Sur cette figure, la clé « k1.example.com » est stockée sur le nœud gérant la zone « example.com ».

Toutefois, comme dit précédemment, le DNS permet de choisir quel serveur stocke quel clé et nous pourrions très bien imaginer une délégation directe de la racine vers un serveur gérant directement la clé « k1.example.com » comme cela est montré sur le Figure 5.13



(a) – Découpage de l'espace de clés (b) – Arbre

FIGURE 5.13 – Exemple montrant qu'un nom « spécifique » peut être délégué par toute zone parente. Ici « k1.example.com » n'est pas gérée par le serveur s'occupant de la zone « example.com ».

Lorsqu'un résolveur DNS veut accéder à une clé, il utilise un processus itératif pour interroger les différents serveurs. Il commence par interroger le serveur racine, qui s'il ne connaît pas la clé demandée, retourne le nom du serveur gérant le sous-espace de clés dont la clé fait partie. Le résolveur va ensuite interroger ce serveur qui pourra à son tour, soit répondre à la requête, soit renseigner le nom d'un serveur gérant un sous-espace de clés de la zone gérée. Un tel processus est illustré dans la Figure 5.14.

Afin que les résolveurs puissent commencer la résolution, l'adresse du serveur racine doit être connue.

Cette façon d'organiser les clés permet de répartir la charge sur différents serveurs. Néanmoins les serveurs proches de la racine sont plus sollicités que ceux des feuilles, nécessitant de les dimensionner plus fortement [VIXIE 1995]. Afin de limiter les requêtes vers les nœuds situés en haut de la hiérarchie, le protocole met en cache non seulement les données demandées par l'utilisateur mais aussi les réponses intermédiaires des différents serveurs [JUNG et al. 2002]. Ainsi, lorsqu'un résolveur cherche la valeur associée à une clé, il peut ne pas commencer à interroger la racine mais un serveur gérant un sous-espace dans lequel la clé fait partie. Concrètement pour résoudre le nom « k1.example.com. », un résolveur peut directement interroger le serveur gérant la zone « example.com. » s'il a appris l'adresse de ce serveur lors d'une précédente résolution (par exemple en résolvant « k2.example.com. »).

Il faut toutefois noter que les enregistrements sont mis en cache pendant une certaine durée (TTL) et que le protocole ne propose pas de mécanisme pour invalider les caches lorsque les données sont modifiées. Il existe toutefois des propositions permettant une consistance forte sur les enregistrements mis en cache [CHEN et al. 2006].

L'utilisation de ce protocole est souvent remis en question par rapport à l'utilisation d'une table de hachage distribuée [COX et al. 2002 ; PARK et al. 2004 ; RAMASUBRAMANIAN et al. 2004 ; GANESAN et al. 2004 ; DOI 2005]. Pourtant, Pappas *et al.* [PAPPAS et al. 2006] ont montré que dans des conditions normales, le DNS et la table de hachage distribuée ont des performances similaires, en fournissant un niveau de disponibilité équivalent. Les différences concernent des aspects non essentiels du protocole : le DNS est plus résistant

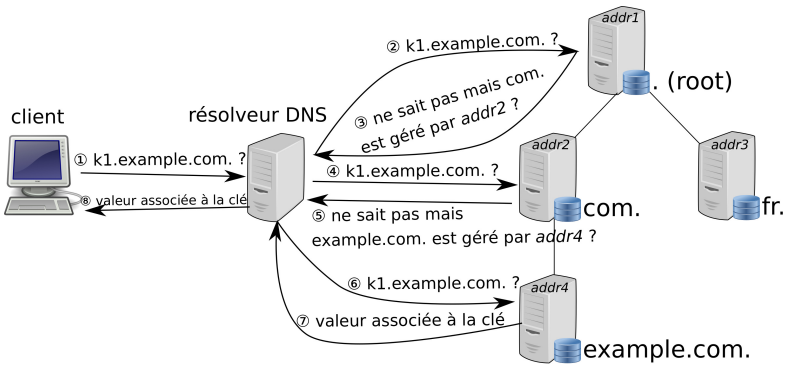


FIGURE 5.14 – Processus de résolution d'un nom DNS (ici « k1.example.com. »). Dans le protocole DNS, le résolveur est le nœud effectuant la résolution.

aux attaques ciblées (notamment avec l'extension DNSSEC [ROSE et al. 2005]) mais la DHT permet de reconfigurer automatiquement le réseau en cas de panne.

Nous noterons que le déploiement du DNS, tout comme des tables de hachages distribuées, se fait sans tenir compte de la topologie physique du réseau. En d'autres mots, par rapport à l'utilisation d'une table de hachage distribuée, dans le DNS, les serveurs ne sont pas placés par tirage d'un identifiant aléatoire. L'administrateur contrôle précisément quel serveur gère quelle portion des clés. Le protocole DNS, à cause de sa flexibilité sur le placement des serveurs, va inspirer la suite de notre travail. Le principal inconvénient de ce protocole est que l'arbre utilisé est un arbre de recouvrement qui ne respecte pas la topologie physique du réseau. Nous cherchons donc quelles sont les méthodes permettant de construire un arbre à partir de la topologie physique du réseau.

5.2.4 Notre proposition de protocole

Nous proposons de distribuer les enregistrements de localisation au sein d'un arbre. De la même façon que dans le protocole DNS, les différents noms sont répartis dans différents serveurs organisés de façon hiérarchique. L'arbre est composé des différents sites de Fog et est parcouru du site courant, jusqu'à la racine. Si la localisation de l'objet n'est pas trouvée à un niveau donné, le site parent est interrogé. Contrairement au DNS où les résolveurs interrogent en premier la racine, notre protocole utilise une approche *bottom-up*. Nous considérons que l'arbre est construit en respectant la topologie physique et les latences dans le but de minimiser le temps d'accès à la localisation d'une donnée. En d'autres mots, le parent de chaque site, est le site le plus proche et interroger ce site est plus rapide qu'interroger le site parent de ce parent. Cette approche permet également de limiter le trafic à une partie restreinte du réseau.

La Figure 5.16 montre l'arbre que nous avons calculé à partir de la topologie du réseau RENATER de la Figure 5.15. Cet arbre a été construit en essayant de réduire les latences réseaux entre la racine et tous les autres nœuds, tout en évitant de placer l'ensemble des sites de Fog directement sous le nœud racine. Notre approche sera détaillée

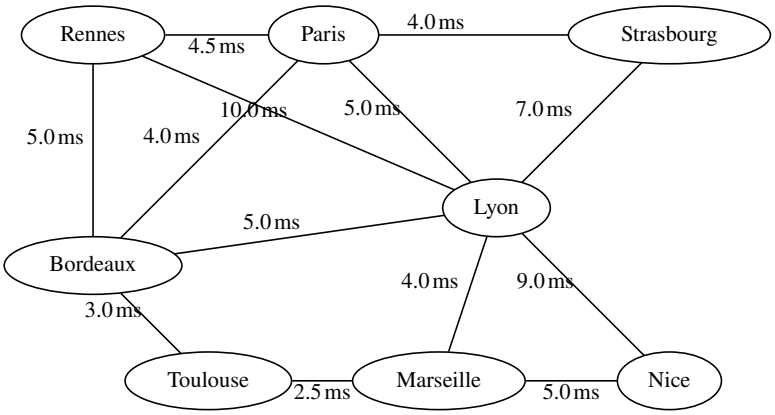


FIGURE 5.15 – Topologie physique simplifiée du Réseau National de télécommunications pour la Technologie l'Enseignement et la Recherche (RENATER).

dans la Section 5.2.6. Dans cette section, cet arbre sera utilisé dans les diagrammes de séquence de la Figure 5.17.

La Figure 5.16 montre également l'organisation des enregistrements de localisation. Les liens entre les sites correspondent à des liens réseaux physiques et chaque nœud est capable de répondre aux requêtes pour les objets stockés dans le sous-arbre. Plus particulièrement, Lyon, à la racine de l'arbre connaît la localisation de tous les objets, c'est pourquoi le nœud racine a été choisi pour sa position centrale dans le réseau. Nous considérons que chaque site est composé de serveurs de stockage et d'un serveur de localisation. Les serveurs de stockage sont responsables de stocker les objets mais se chargent aussi de localiser et télécharger les objets demandés par les clients, pour lesquels ils ne stockent pas de réplica. Les serveurs de localisation quant à eux, ne font que stocker des enregistrements associant un nom d'objet à une liste de sites sur lequel un réplica se trouve.

La Figure 5.16 montre aussi le contenu initial des serveurs de localisation. Par exemple, l'enregistrement **.paris* définit la localisation par défaut de tous les objets suffixés avec *.paris*. Ainsi, lorsqu'un client écrit un objet, comme illustré dans la Figure 5.17(a), un suffixe correspondant au nom du site est ajouté à la fin du nom. Ce type de délégation générique permet de ne pas avoir à mettre à jour les métadonnées lorsque l'objet est écrit pour la première fois et n'a pas été encore répliqué sur les autres sites. Nous posons l'hypothèse que dans une infrastructure de Fog Computing, la plupart des objets sont écrits localement et ne sont que rarement lus depuis un site distant. Cette approche permet de réduire significativement le trafic inter-sites mais aussi le nombre d'enregistrements à stocker sur les serveurs de localisation. Néanmoins, s'appuyer seulement sur le suffixe d'un objet pour déterminer l'emplacement de ces réplicas n'est pas suffisant. Bien qu'un nœud de stockage situé à Paris pourrait être contacté directement, il pourrait exister un réplica plus proche dans le réseau. C'est pourquoi nous utilisons les suffixes uniquement pour limiter le nombre de messages lors de l'écriture de nouveaux objets.

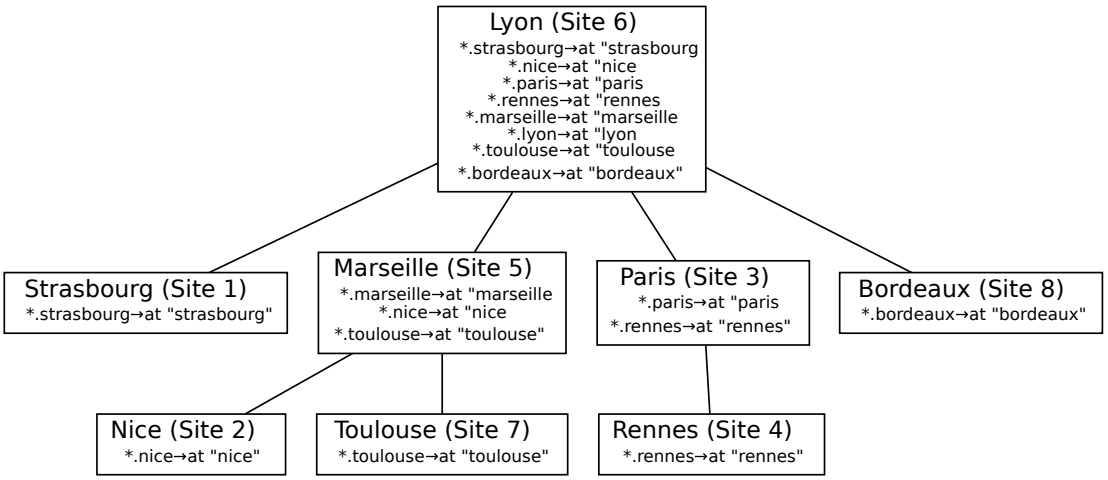
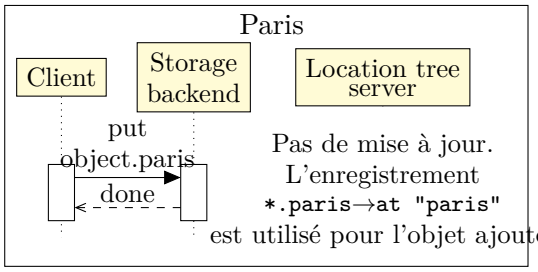


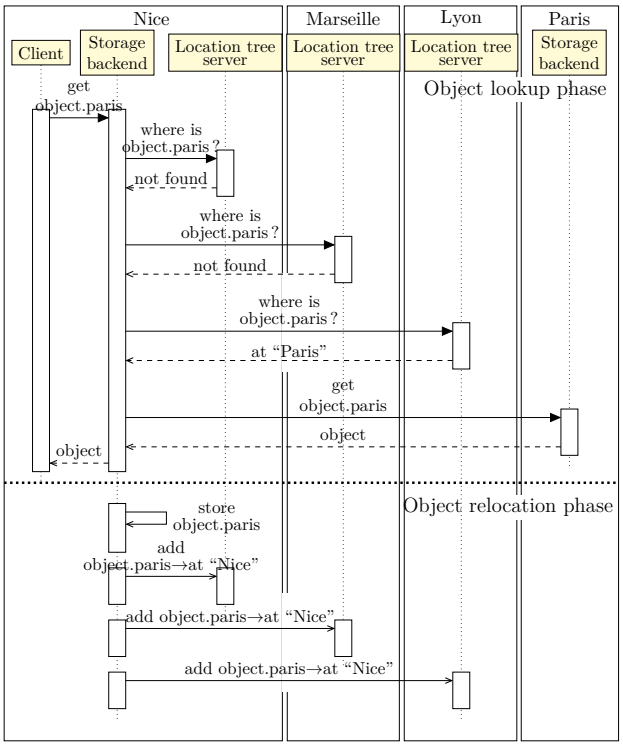
FIGURE 5.16 – Arbre calculé avec notre algorithme, montrant le contenu initial des serveurs de localisation. Chaque site possède également des serveurs de stockage qui ne sont pas représentés.

Nous devons également préciser que l'opération d'écriture ne concerne que des nouveaux objets puisque nous avons considéré l'hypothèse d'immuabilité des réplicas. De cette façon, notre système n'a pas à gérer les mises à jour d'objets.

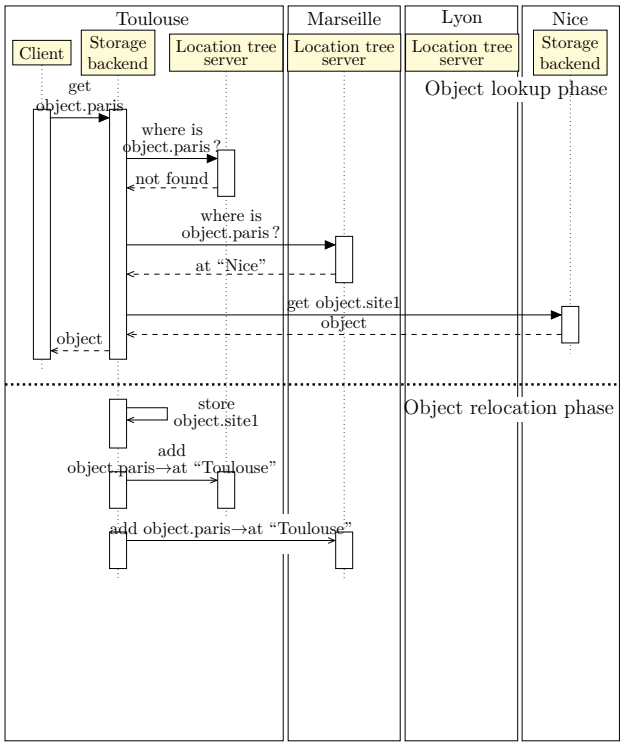
La Figure 5.17(b) montre le processus de lecture, lorsqu'un client situé à Nice souhaite accéder à l'objet stocké à Paris. Le client interroge d'abord son serveur de stockage local pour vérifier si l'objet n'est pas présent localement. Si cela n'est pas le cas, le serveur de localisation, présent sur le site, est interrogé. Si celui-ci ne connaît pas la localisation de l'objet, alors le serveur parent est interrogé (*i.e.*, Marseille), et ainsi de suite, jusqu'à ce que la localisation soit déterminée. Dans le pire des cas, la localisation n'est trouvée qu'une fois le serveur racine atteint (*i.e.*, Lyon). Une fois que la localisation est trouvée, un nouveau réplica est créé et les enregistrements de localisation sont mis à jours de façon asynchrone. Le serveur de stockage ayant accédé à l'objet met à jour les serveurs stockant la localisation. Seuls les serveurs du serveur local jusqu'au serveur sur lequel la localisation a précédemment été trouvée sont mis à jour. Par exemple sur la Figure 5.17(b), Nice localise un objet en accédant au site de Lyon. Une fois le nouveau réplica créé à Nice, les serveurs de localisation situés à Nice, Marseille et Lyon doivent être mis à jour. Cela permet non seulement de limiter le nombre de messages pour mettre à jour la localisation mais permet aussi aux sites situés dans le sous-arbre d'accéder à ce nouveau réplica, situé plus proche que le réplica qui a été accédé précédemment. Par exemple sur la Figure 5.17(c), avoir relocalisé l'objet à Nice et mis à jour les serveurs de localisation permet au site de Toulouse, non seulement de localiser l'objet en interrogeant le serveur situé à Marseille, plus proche que Lyon, mais aussi de récupérer l'objet depuis le réplica stocké à Nice, accessible avec une plus faible latence que Paris. La Figure 5.18 montre l'état de l'arbre une fois qu'un réplica de l'objet a été créé à Nice et que les



(a) – Écriture d'un objet sur le site local (Paris).



(b) – Lecture depuis Nice d'un objet stocké à Paris.



(c) – Lecture depuis Toulouse d'un objet précédemment relocalisé à Nice.

FIGURE 5.17 – Diagrammes de séquence montrant le processus d'écriture lorsqu'un client écrit un objet à Paris (a) et le processus de lecture lorsque l'objet est lu depuis Nice (b) et Toulouse (c).

nouveaux enregistrements de localisation ont été créés.

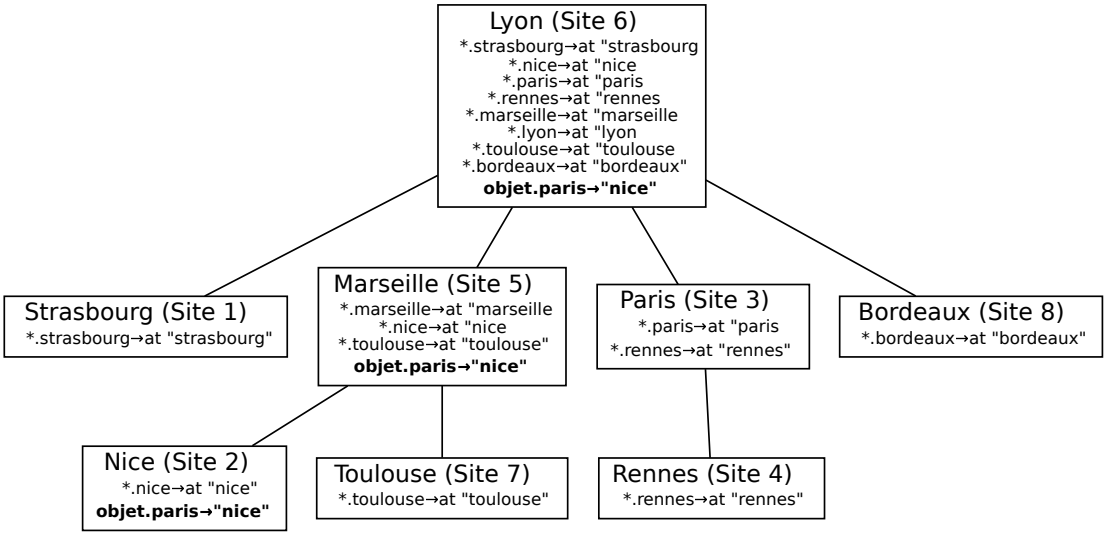


FIGURE 5.18 – Arbre montrant les enregistrements de localisation une fois qu'un nouveau réplica de l'objet a été créé à Nice.

La Figure 5.17(c) montre les échanges lorsque Toulouse demande à accéder à l'objet pour lequel un nouveau réplica vient d'être créé à Nice. Le processus de lecture est le même que celui précédemment décrit mais cette fois-ci, le réplica situé à Nice est accédé grâce à la localisation trouvée à Marseille. Le serveur situé à la racine de l'arbre n'est ni mis à jour, ni interrogé.

Nous mettons également l'accent sur le fait que la création du réplica à Nice n'induit pas une modification du nom de l'objet. Toulouse accède toujours à l'objet suffixé avec *.paris* mais l'arbre permet de ne pas accéder au réplica situé à Paris mais à un réplica plus proche. Les suffixes n'ont pas de signification dans le processus de lecture.

Cette approche a plusieurs avantages. Premièrement, les objets qui ne sont jamais accédés ne génèrent pas de trafic inter-sites. Deuxièmement, plus un objet est accédé, plus sa localisation pourra être trouvée proche du site souhaitant accéder à l'objet, réduisant d'autant le temps d'accès. Enfin, si l'utilisateur se déplace sur une petite portion du réseau, l'objet pourra être localisé sans jamais interroger le serveur racine.

Comme chaque nœud stocke la localisation des objets stockés dans son sous-arbre, cela signifie qu'en cas de partitionnement du réseau, il est toujours possible pour chacun d'entre eux de localiser les données stockées au sein de la partition.

Pour conclure, nous pensons que notre protocole est plus adapté que l'utilisation d'une table de hachage distribuée car *(i)* la localisation des objets est trouvée sur le chemin vers ceux-ci : dans le pire des cas, en parcourant la moitié du réseau, et *(ii)* les réplicas de localisation sont créés et distribués en fonction des accès. Autrement dit, si un objet n'est jamais accédé

Finalement, en plus de réduire les temps d'accès, placer des réplicas de localisa-

tion proche des réplicas de données permet de rendre les objets accessibles en cas de partitionnement du réseau.

5.2.5 Algorithmes pour la génération d'arbres couvrants

L'arbre sur lequel repose notre protocole doit être construit avec le plus grand soin. En effet, la structure de l'arbre détermine, le nombre de sauts maximal pour localiser un objet, les opportunités pour les nœuds de pouvoir créer de nouveaux réplicas de localisation, mais également la latence réseau pour atteindre chacun des nœuds. Nous explorons dans cette partie les algorithmes qui permettent de construire un arbre couvrant à partir de la topologie physique du réseau avant de proposer notre approche qui prend en compte les spécificités de notre protocole.

Arbre couvrant de poids minimal

L'arbre couvrant de poids minimal consiste à construire un arbre dont la somme du poids des arêtes est la plus petite possible [Kershenbaum et al. 1972]. Deux algorithmes sont principalement utilisés. L'algorithme de Kruskall et l'algorithme de Prim. L'algorithme de Kruskall fonctionne en triant les arêtes par poids croissants et en insérant dans l'arbre celles de poids le plus faible qui ne créent pas de cycle [Kruskal 1956]. L'algorithme de Prim fait la même chose mais en découvrant le graphe au fur et à mesure. L'idée est d'ajouter le nœud qui se connecte à l'arbre en cours de construction grâce à l'arête de poids le plus faible [Prim 1957].

Les arbres couvrants minimisent la distance entre chaque nœud et son parent mais ne minimisent pas la distance totale entre chaque nœud et la racine de l'arbre. D'ailleurs, tout nœud de l'arbre peut jouer le rôle de racine sans changer le poids total de ce dernier.

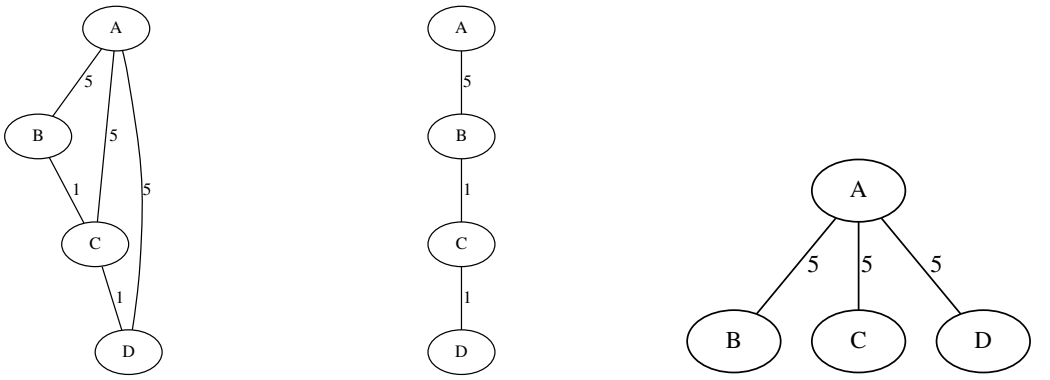
Arbre des plus courts chemins

L'arbre des plus courts chemins est l'arbre qui à partir d'un nœud source permet de joindre tous les autres sommets en utilisant les chemins les plus courts. Un algorithme traditionnel pour calculer un tel arbre est l'algorithme de Dijkstra [Dijkstra 1959].

L'arbre total minimise indépendamment la distance entre chaque nœud et la racine, ce qui ne minimise pas le poids total de l'arbre : la somme du poids des arêtes de l'arbre n'est pas minimal. Par exemple, sur la Figure 5.19 le chemin de A vers D a un poids de 5 alors que dans un arbre couvrant minimal de même racine, ce dernier a un poids de 7.

Parce que l'algorithme Dijkstra génère des arbres très « plats » et que l'algorithme de Prim génère des arbres dégénérés, des algorithmes comme AHHK (du nom de ses auteurs Alpert Hu Huang et Kahng) [Alpert et al. 1993] permettent de générer un arbre qui est un compromis entre l'algorithme de Dijkstra et l'algorithme de Prim. L'algorithme proposé dans ce travail est similaire à celui de Dijkstra mais la fonction de coût est modifiée.

Nous désignons par $d(x, y)$ la distance entre un nœud x et un nœud y. Pour nous la distance $d(x, y)$ est la latence réseau entre le nœud x et le nœud y mais nous pourrions



(a) – Graphe

(b) – Arbre couvrant de poids minimal

(c) – Arbre des plus courts chemins

FIGURE 5.19 – Exemple de graphe avec un arbre couvrant de poids minimal (a) et un arbre des plus courts chemins (b) en considérant A comme le nœud source.

également prendre en compte les débits ou la capacité de stockage des nœuds. Dans l'algorithme de Dijkstra, les nœuds sont sélectionnés les uns après les autres selon leur distance avec le nœud source. Lorsque le nœud x est sélectionné car la distance $d(n\triangleleft ud\_source, x)$ est la plus petite, nous vérifions pour chaque autre nœud si le chemin du nœud source à ce nœud a une distance plus faible en passant par le nœud x. L'équation 5.1 montre la fonction de coût utilisée pour évaluer la distance entre le nœud source (racine de l'arbre) et un nœud y, lorsque le nœud x est sélectionné.

$$f_c = d\,(\text{racine}, y) < d\,(\text{racine}, x) + d\,(x, y) \tag{5.1}$$

Alpert *et al.* proposent d'introduire un facteur c (compris entre 0 et 1) et d'utiliser la fonction d'évaluation proposée dans l'Équation 5.2. L'idée est de réduire artificiellement la distance jusqu'au parent du nœud évalué, de façon à favoriser un placement plus profond des nœuds dans l'arbre.

$$f_c = d\,(\text{racine}, y) < (c \times d\,(\text{racine}, x)) + d\,(x, y) \tag{5.2}$$

Lorsque la valeur de c est égale à 1, l'algorithme est similaire à l'algorithme de Dijkstra, lorsqu'elle est égale à 0, un arbre couvrant de poids minimum est généré. Entre ces deux valeurs, l'arbre généré est un compromis entre l'arbre des plus courts chemins et l'arbre couvrant de poids minimal.

Optimisation linéaire

La construction d'un arbre à partir de plusieurs points d'un graphe peut également être considérée comme un problème d'optimisation, de type « Problème de l'emplacement d'installations » (« Facility Location Problem ») qui consiste à placer des entrepôts au plus près des clients tout en ne s'éloignant pas trop des usines alimentant ces entrepôts. Ce problème est un problème NP-difficile et plusieurs heuristiques ont été proposées, telles que, *K-means*, *K-median* ou *K-critical* [Qiu et al. 2001].

Le problème peut également être vu comme un problème de Steiner, qui consiste à créer un arbre à partir des nœuds feuilles [Lin et al. 1999]. Ce problème est légèrement différent de celui présenté précédemment, dans la mesure où les nœuds du graphe topologique n'ont pas une position « fixée » à l'avance. Ce problème doit plutôt être utilisé lorsque nous cherchons à placer physiquement les sites de Fog.

5.2.6 Notre approche de construction de l'arbre

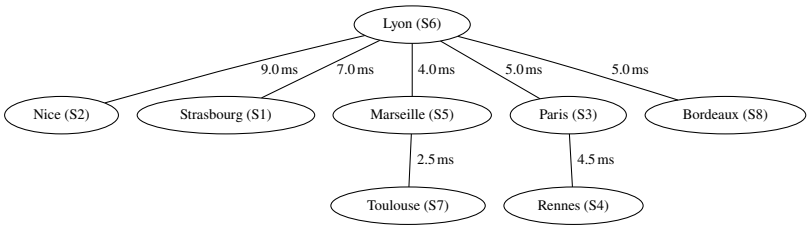
Comme vu dans la section précédente, la latence totale pour localiser un objet augmente avec la profondeur de l'arbre. Par exemple, dans la Figure 5.16, lorsqu'un nœud trouve la localisation à Lyon, la latence totale est la latence entre Nice et Marseille pour demander la localisation à Marseille à laquelle s'ajoute la latence entre Nice et Lyon pour interroger le serveur situé à Lyon. Le lien entre Nice et Marseille est donc sollicité deux fois. Il est donc important de construire l'arbre avec soin si nous voulons minimiser les temps d'accès.

Nous proposons de construire un arbre des plus courts chemins, qui minimise la latence totale entre chaque nœud et la racine. Au lieu d'utiliser la fonction d'évaluation originale de l'algorithme de Dijkstra, nous proposons notre propre fonction d'évaluation (Équation 5.3) qui prend en compte la profondeur des nœuds. En effet, dans notre protocole itératif, plus un lien est profond, plus il est sollicité.

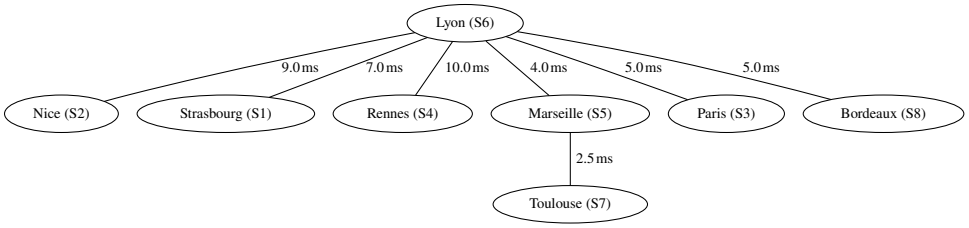
$$f_c = \left(\sum_{i=racine}^{parent(\text{nœud})} d\left(i, parent\left(\text{i}\right)\right) \times profondeur\left(i\right) \right) + d\left(parent\left(\text{nœud}\right), \text{nœud}\right) \times profondeur\left(\text{nœud}\right) \tag{5.3}$$

Par exemple dans la Figure 5.16, le site de Nice envoie d'abord une requête sur le site de Marseille avant d'en envoyer une à Lyon. Par conséquent, le lien réseau entre Nice et Marseille est sollicité deux fois tandis que le lien de Marseille à Lyon n'est sollicité qu'une seule fois. C'est cette façon d'exécuter les requêtes qui nous a conduit à modifier la fonction d'évaluation utilisée par l'algorithme de Dijkstra afin de favoriser les liens à faible latence en bas de l'arbre.

Dans l'algorithme de Dijkstra, le nœud source correspond à la racine de l'arbre et doit être explicitement spécifié par l'utilisateur lors de l'exécution de l'algorithme. Il n'est pas automatiquement déterminé par l'algorithme. Nous proposons donc considérer successivement chaque nœud comme nœud source et de générer l'arbre correspondant. Une fois chaque arbre généré, nous sélectionnons l'arbre ayant le poids le plus faible.



(a) – Dijkstra.



(b) – « Notre métrique ».

FIGURE 5.20 – Arbres générés avec l'algorithme de Dijkstra standard (a) et en utilisant notre fonction de coût (b).

Nous supposons que la topologie est stable, de manière à n'effectuer le calcul de l'arbre qu'une seule fois.

Exécuter l'algorithme de Dijkstra avec notre fonction de coût et notre façon de sélectionner le nœud racine sur le graphe donné dans la Figure 5.15 permet de construire l'arbre de la Figure 5.20(b). Bien que cet arbre optimise la latence totale, en prenant en compte la latence des « sauts » intermédiaires, l'arbre généré est « plat », ce qui ne permet pas de bénéficier de la création de nouveaux réplicas de localisation. En effet, dans une telle configuration, la racine est directement contactée sans avoir eu la possibilité d'interroger des nœuds intermédiaires.

Afin de générer un arbre plus profond, nous introduisons un mécanisme similaire à celui proposé par AHHK [ALPERT et al. 1993]. Nous connectons un site à une nouvelle position dans l'arbre si le site est placé à une profondeur plus élevée que sa profondeur actuelle et si la latence totale (mesurée par l'équation 5.3) est meilleure ou dégradée d'un facteur inférieur à c. Même si la latence pour atteindre tous les ancêtres jusqu'à la racine est plus élevée, un nœud plus profond dans l'arbre, avec plus d'ancêtres, a de plus grandes chances de trouver un enregistrement de localisation parmi eux.

La Figure 5.16 de la Section 5.2.4 montre l'arbre calculé avec cet algorithme en utilisant $c = 1,2$ et la topologie du réseau RENATER de la Figure 5.15. Cet arbre essaie de minimiser la latence totale jusqu'à la racine mais permet aussi aux nœuds de bénéficier de la relocalisation des enregistrements de localisation.

Afin de montrer que cet arbre est meilleur que l'arbre optimal (celui calculé avec l'équation 5.3, sans dégrader les latences en tenant compte de la valeur de c), nous

calculons une métrique (dans l'équation 5.4) dans laquelle les latences sont pondérées de la probabilité $p(j)$ de trouver la localisation sur le nœud j. De cette façon, nous prenons en compte le fait qu'un nœud profond dans l'arbre a plus de chances de trouver la localisation de l'objet avant d'atteindre la racine qu'un nœud situé directement sous la racine.

$$w_{\text{arbre}} = \sum_{i \in \text{nœuds}} \sum_{j=\text{nœud}}^{racine} d(i,j) \times p(j) \tag{5.4}$$

Cette métrique ne peut pas être utilisée directement dans l'algorithme de Dijkstra car la probabilité $p(j)$ dépend du nombre d'enfants qu'un nœud possède. Or, lorsque nous plaçons un nœud dans l'arbre, nous ne connaissons pas encore cette valeur.

Pour illustrer l'utilisation de cette métrique, le nœud Marseille de la Figure 5.20(b) a un poids de $4,0 \times \frac{6}{7} + 0 \times \frac{1}{7} = 3,43$. En effet, lorsque nous accédons à un objet depuis Marseille, si cet objet a été lu depuis Toulouse, la localisation est trouvée localement ($0\,\text{ms} \times \times \frac{1}{7}$). En revanche, si l'objet a été lu des 6 autres sites, la localisation ne peut être trouvée qu'à Lyon, atteignable en 4,0 ms ($4,0\,\text{ms} \times \times \frac{6}{7}$).

En d'autres termes, le poids total de l'arbre de la Figure 5.20(b) se calcule avec la formule donnée dans l'équation 5.5. De la même façon, l'équation 5.6 donne le poids total de l'arbre généré dans le cas où $c = 1.2$ (Figure 5.16).

$$\begin{aligned} w &= \left(9,0 \times \tfrac{7}{7}\right) + \left(7,0 \times \tfrac{7}{7}\right) + \left(10,0 \times \tfrac{7}{7}\right) + \left(4,0 \times \tfrac{6}{7}\right) + \left(5,0 \times \tfrac{7}{7}\right) + \left(5,0 \times \tfrac{7}{7}\right) + \left(2,5 \times \tfrac{1}{7}\right) \\ &+ \left((2,5+4,0) \times \tfrac{6}{7}\right) \\ w &\approx 45,4 \end{aligned} \tag{5.5}$$

$$\begin{aligned} w &= \left(7,0 \times \tfrac{7}{7}\right) + \left(4,0 \times \tfrac{5}{7}\right) + \left(5,0 \times \tfrac{6}{7}\right) + \left(5,0 \times \tfrac{7}{7}\right) + \left(5,0 \times \tfrac{2}{7}\right) + \left((5,0+4,0) \times \tfrac{5}{7}\right) \\ &+ \left(2,5 \times \tfrac{2}{7}\right) + \left((2,5+4,0) \times \tfrac{5}{7}\right) + \left(4,5 \times \tfrac{1}{7}\right) + \left((4,5+5,0) \times \tfrac{6}{7}\right) \\ w &\approx 41,1 \end{aligned} \tag{5.6}$$

Notre arbre utilisant un algorithme inspiré de AHHK a un poids plus faible que l'arbre optimal calculé précédemment.

5.2.7 Surcoût de notre approche

Un inconvénient de notre approche par rapport à une table de hachage distribuée est le nombre de messages envoyés pour localiser et mettre à jour les enregistrements de localisation. Dans cette section, nous évaluons cette complexité mais aussi la complexité pour le calcul de l'arbre. Dans notre comparaison, nous ne considérons pas l'accès au serveur de localisation situé sur le site local car ce dernier est accessible avec une très faible latence et ne génère pas de trafic réseau inter-sites. Nous posons donc l'hypothèse que cet accès n'a pas d'impact sur les performances.

Dans les Figures 5.17(b) et 5.17(c), tous les serveurs de localisation contactés pour localiser un objet sont mis à jour une fois le nouveau réplica créé. Dans le pire des cas, $hauteur(arbre)$ messages de mises à jour sont envoyés ($O(n-1)$ avec n, le nombre de sites). Dans la DHT, le nombre de messages de mise à jour est égal au nombre de réplicas

de localisation, qui est une constante ($O(1)$). Notre approche est donc dans beaucoup de cas, plus coûteuse qu'une table de hachage distribuée.

Le nombre de messages de mise à jour varie selon les mouvements de données dans notre approche. Nous avons donc besoin de considérer le nombre total de messages émis, lorsqu'un objet est accédé successivement depuis l'ensemble des sites. Dans notre approche, le nombre total de messages pour trouver la localisation est égal au nombre d'arêtes dans l'arbre (*i.e.*, le nombre de sites moins un), soit n messages. Dans la DHT, lorsqu'il n'y a pas de réplication, une requête nécessitant $log(n)$ sauts est envoyée à chaque accès (avec n le nombre de nœuds dans le réseau), générant un trafic égal à $n \times log(n)$ messages (soit $k \times n \times log(n)$ messages lorsque chaque clé est répliquée k fois). De ce point de vue, notre protocole est plus économe en messages que l'utilisation d'une table de hachage distribuée.

Bien que dans la DHT, le calcul des tables de routage génère du trafic réseau, le calcul de l'arbre utilisé dans notre approche a un coût qui ne doit pas être négligé. La complexité de notre calcul est de l'ordre de $O(n^3)$ où n est le nombre de sites dans la topologie, puisque nous exécutons n fois l'algorithme de Dijkstra qui a une complexité de l'ordre de $O(n^2)$. Cela signifie que recalculer un arbre à chaque fois qu'un site est ajouté ou quitte le réseau n'est peut-être pas optimal mais que cela peut être fait régulièrement, de manière proactive.

5.2.8 Conclusions

Utiliser un arbre dans un contexte multi-sites n'est pas une approche nouvelle et a déjà été proposée à de nombreuses reprises [Mockapetris 1987 ; Plaxton et al. 1996 ; Karger et al. 1997]. La particularité de notre approche est que notre arbre est construit en tenant compte de la topologie du réseau d'une part et d'autre part, que la propagation des informations se fait des feuilles de l'arbre vers la racine, Cela permet de confiner le trafic réseau en essayant d'interroger des sites voisins en premier tout en garantissant que la localisation d'un objet soit trouvée (contrairement aux approches par inondation). Cela permet également d'améliorer le comportement en cas de partitionnement du réseau puisque dans ce cas, l'ensemble des objets situés dans le sous-arbre déconnecté restent accessibles.

Toutefois, le coût pour calculer l'arbre, et le manque de support de la dynamique du réseau sont encore des points sur lesquels des améliorations sont nécessaires. En effet, il ne semble pas concevable de recalculer entièrement l'arbre et de déplacer les enregistrements de localisation à chaque fois qu'un site est ajouté ou supprimé du réseau.

Les caractéristiques de notre approche sont résumées dans la Figure 5.21.

5.3 Conclusion

Dans ce chapitre, nous avons présenté deux stratégies permettant de confiner le trafic réseau et de limiter les trafics inter-sites, impactant les temps d'accès et compromettant la disponibilité des données en cas de partitionnement du réseau. Nous avons considéré deux problématiques. La première fut de limiter les trafics inter-sites lors de l'accès à

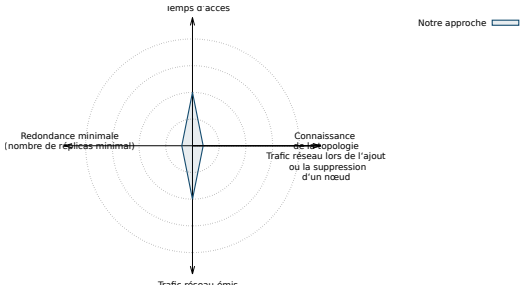


FIGURE 5.21 – Diagramme en étoile résumant les caractéristiques de notre approche.

des données stockées localement sur le site tandis que la seconde consiste à confiner les trafics inter-sites lors d'accès à des données stockées sur un site distant.

Pour la première problématique, nous avons proposé de grouper les nœuds d'un même site en déployant un *Scale-Out NAS* commun à tous les nœuds du site. Ce couplage d'IPFS avec un *Scale-Out NAS* permet d'accéder aux données stockées localement sur le site de Fog, sans échanges avec les sites distants. Cela permet non seulement d'améliorer les performances mais aussi de rendre les données du site disponibles en cas de partitionnement du réseau entre les sites.

Pour la seconde problématique, nous avons remplacé la table de hachage distribuée par un protocole de gestion des localisations qui considère la topologie du réseau physique. Nous avons proposé une approche reposant sur un arbre couvrant. Cette approche permet de localiser les objets en interrogeant en premier les sites « proches » du site courant, de façon à confiner le trafic réseau. Placer les réplicas de localisation proche des réplicas de données permet non seulement de réduire les temps de localisation mais aussi les temps d'accès, en téléchargeant les données depuis l'un des sites les plus proches. Dans le chapitre suivant, nous proposons une évaluation expérimentale de ces deux approches.

6

Évaluation expérimentale des approches proposées

Sommaire

Dans ce chapitre, nous évaluons expérimentalement les deux adaptations que nous avons proposées dans le chapitre précédent afin de minimiser les échanges inter-sites dans le cadre d'un partage de données dans un environnement de Fog Computing. Ces expérimentations ont été réalisées sur la plateforme d'évaluation Grid'5000. Dans un premier temps, nous montrons que l'utilisation d'un *Scale-Out NAS* couplé à IPFS permet de réduire les temps de lecture d'environ 34 % tout en évitant que du trafic réseau soit échangé entre les sites lorsque l'objet accédé est stocké localement sur un site. Dans un second temps, nous évaluons notre approche en arbre afin de stocker la localisation des objets. Nous montrons non seulement que cette approche permet de réduire le temps nécessaire à la localisation d'un objet mais également qu'elle permet de localiser le réplica le plus proche. Enfin, nous donnons des éléments afin de construire un environnement d'évaluation plus réaliste, en interconnectant les plateformes Grid'5000 et FIT/IoT-Lab. L'objectif visé est d'offrir la possibilité d'utiliser un site de Fog émulé sur Grid'5000 à partir de véritables objets connectés hébergés sur la plateforme IoT-Lab.

6.1 Évaluation du couplage IPFS et d'un *Scale-Out NAS*

Nous proposons dans cette section d'évaluer notre couplage d'IPFS avec un système de fichiers distribué, déployé localement et indépendamment sur chaque site. Ce travail a fait l'objet d'une publication à la conférence ICFEC 2017 [Confais et al. 2017b]

Implémentation

Afin qu'IPFS stocke les objets dans un système de fichiers distribué, nous avons réalisé quelques modifications dans le code source. Premièrement, nous avons déplacé le dossier utilisé par IPFS pour stocker les objets, sur le point de montage utilisé par le système de fichiers distribué. Cela a dû être fait sans déplacer la base de donnée locale utilisée par chaque nœud. Chaque nœud utilise sa propre base de données dans laquelle il stocke son état, son identifiant et sa table de routage pour la table de hachage distribuée. Les objets sont quant à eux stockés dans le *Scale-Out NAS* afin de rendre ceux-ci disponibles pour l'ensemble des nœuds du site.

Deuxièmement, nous avons supprimé le cache utilisé par le module *blockstore* d'IPFS, s'occupant de stocker les objets dans le système de fichiers local. Le cache est conçu pour garder en mémoire le fait qu'un objet existe ou non. Cette modification est obligatoire pour forcer le nœud à vérifier la présence de l'objet dans le *Scale-Out NAS* avant d'utiliser la DHT. En effet, lorsque l'on demande un objet à un nœud qui ne l'a pas stocké avant, la valeur dans ce cache est que l'objet n'existe pas et cela a pour conséquence, que le nœud accède à la DHT avant de vérifier si un nœud du site n'a pas stocké l'objet dans le système de fichiers distribué.

Enfin, nous avons modifié la façon dont IPFS émet les requêtes vers les autres nœuds. Par défaut, chaque fois qu'IPFS cherche à télécharger un objet stocké sur un site distant, il recherche la localisation de l'objet dans la DHT puis contacte les nœuds stockant un réplica. Cependant, au lieu de ne demander uniquement l'objet souhaité

au nœud distant, IPFS demande tous les objets dont il a besoin et pour lesquels le téléchargement n'est pas terminé. Cela augmente le trafic réseau puisque certaines pièces d'objets sont demandées à plusieurs nœuds simultanément. La modification consiste donc à limiter les demandes d'IPFS aux seuls nœuds spécifiés dans la DHT. Au total, les modifications réalisées correspondent à environ 250 lignes de code dans le langage de programmation Go. Le code source est disponible en ligne, à l'adresse suivante : https://github.com/bconfais/go-ipfs/tree/common_backend_v1a.

6.1.1 Matériels et méthodes

Notre évaluation a été réalisée en utilisant trois architectures logicielles différentes, toutes déployées sur la plateforme Grid'5000 [Balouek et al. 2013] :

1. IPFS dans sa configuration par défaut, déployé dans un Cloud ;
2. IPFS dans sa configuration par défaut, déployé dans un environnement Fog/Edge ;
3. IPFS couplé avec un *Scale-Out NAS* indépendant, déployé dans un environnement Fog/Edge.

Nous avons choisi d'utiliser RozoFS [Pertin et al. 2014] comme *Scale-Out NAS*. RozoFS est une solution *open-source* respectant le standard POSIX et obtenant un bon débit, que ce soit pour les accès séquentiels ou aléatoires, en lecture ou en écriture. À la manière de beaucoup de *Scale-Out NAS*, RozoFS s'appuie sur un serveur de métadonnées centralisé pour localiser les données. La distribution des données sur les différents serveurs de stockage est réalisée à l'aide d'un code à effacement Mojette, rendant le système tolérant aux pannes (ici 2 projections Mojette sur 3 sont nécessaires pour reconstruire la donnée). Les données sont découpées en blocs (4 Ko par exemple). Pour chaque bloc, le serveur de métadonnées associe un certain nombre de serveurs de stockage (3 dans notre cas). Pour écrire, le client calcule 3 projections du bloc et écrit chacune des projections sur chacun des serveurs de stockage spécifié par le serveur de métadonnées. Pour la lecture, dans notre cas, seulement 2 projections sont récupérées par le client qui est ensuite capable de reconstruire le bloc de données. Plus de détails sont donnés dans l'article de Pertin *et al.* [Pertin et al. 2014]. Nous mettons l'accent sur l'utilisation de ce code à effacement puisque l'utilisation de RozoFS génère non seulement un surplus de calcul pour générer les projections mais également un surplus d'écriture de 50 % en taille. Cela permet à RozoFS de tolérer la panne d'un seul serveur de stockage sans rendre les données indisponibles mais ces surcoûts pourraient impacter les temps d'accès. C'est pourquoi, dans la suite de ce travail, nous devons non seulement évaluer les gains potentiels vis-à-vis de la table de hachage distribuée mais également le surcoût de ce couplage. Il est également possible de répliquer le serveur de métadonnées au sein d'un site pour éviter les points uniques de défaillance. Enfin, nous précisons que notre proposition s'adapte à tout type de *Scale-Out NAS* comme GlusterFS [Davies et al. 2013], Lustre [Donovan et al. 2003] ou même CephFS [Weil et al. 2007]. Nous avons choisi RozoFS car nous avons une bonne connaissance de ses mécanismes internes, facilitant l'analyse de nos résultats.

Nous considérons que chaque serveur de stockage a les deux rôles : être un serveur de stockage IPFS mais aussi un nœud de stockage RozoFS (colocalisation). Pour éviter les biais liés aux performances du système de fichiers sous-jacent, nous utilisons un `tmpfs` comme système de stockage de bas niveau et effaçons les caches après chaque opération. Cela permet non seulement d'enlever des briques logicielles qui pourraient influer sur les temps d'accès : les temps d'accès ne dépendent que du couplage RozoFS/IPFS mais aussi de faciliter la reproductibilité de l'expérimentation (pas de données en cache qui pourraient améliorer significativement une lecture).

La topologie que nous évaluons correspond à celle illustrée dans la Figure 5.1. Celle-ci est composée de 3 sites, chacun composé de 6 nœuds : 4 serveurs de stockage, un serveur de métadonnées pour RozoFS et un client. Le déploiement Cloud d'IPFS est composé de 12 serveurs (le même nombre que dans les déploiements en mode Fog). La topologie ne change pas ce qui nous évite de gérer le *churn*.

Les latences réseaux entre les différents nœuds ont été choisies de la même manière que dans l'expérimentation du Chapitre 4 :

- $L_{Fog} = 10ms$ est la latence entre les clients et le site de Fog auquel ils appartiennent ;
- $L_{Core} = 50ms$ est la latence entre les sites de Fog ;
- $L_{Cloud} = 200ms$ est la latence pour atteindre une plateforme de Cloud [Souza Couto et al. 2014] ;
- $L_{Site} = 0.5ms$ est la latence entre les serveurs d'un même site de Fog mais aussi la latence entre les différents serveurs d'une infrastructure de Cloud Computing.

Les latences sont émulées grâce à la commande `tc` fournie par l'outil *Linux Traffic Control Utility*. Nous n'avons pas modifié le débit des liens qui est de $10Gbps$ et nous rappelons, comme dit dans le Chapitre 4, qu'augmenter la latence a un impact non négligeable sur le débit atteignable par les connexions TCP. En mesurant le débit en utilisant l'outil *iperf*, nous obtenons un débit de 2,84 Gbps entre un client et son site de Fog. 9,41 Gbps, entre les serveurs de stockage d'un même site et 533 Mbps entre les sites. Nous mesurons un débit de 250 Mbps pour atteindre une plateforme de Cloud. Les expérimentations ont été effectuées dans un objectif de reproductibilité [Imbert et al. 2013]. Pour cela nous avons écrit un script de 800 lignes en Python qui déploie automatiquement IPFS sur la plateforme Grid'5000, affecte les latences aux liens réseau et exécute l'expérimentation. Le code de ce script ainsi que les résultats des expérimentations sont disponible à l'adresse suivante : `https://github.com/bconfais/benchmark/tree/master/FEC2017`.

Nous avons mesuré les temps d'accès, en utilisant notre propre script ainsi que la quantité de trafic inter-sites échangé en utilisant `iptables`. Les temps d'accès correspondent aux temps pour exécuter une opération de lecture ou d'écriture, du point de vue du client. Nous avions précédemment effectué ces tests avec Yahoo Cloud System Benchmark (YCSB) [Cooper et al. 2010], en utilisant un module IPFS que nous avions développé [Confais et al. 2017d]. Cependant, YCSB prend en compte dans la mesure du

temps d'accès, le temps de génération d'un objet à partir d'octets aléatoires (lus depuis `/dev/urandom`). Ce temps peut être significatif au regard de certains temps d'écriture, particulièrement pour les objets de taille importante (10 Mo). C'est pourquoi, nous avons choisi d'utiliser notre propre script dans lequel nous ne mesurons que le temps mis pour exécuter l'instruction d'écriture ou de lecture d'un objet. Également, comme cela était le cas dans le Chapite 4, le client choisit aléatoirement le nœud IPFS du site auquel il transmet la requête. Chaque scénario est exécuté 10 fois pour la consistance des résultats.

6.1.2 Écriture/Lecture depuis le site local

Dans cette Section, nous discutons des résultats obtenus lorsque des clients écrivent et lisent simultanément des objets stockés sur le site local. Nous comparons également les résultats avec le cas où les objets sont stockés dans une infrastructure de type Cloud. Notre objectif est de comprendre comment le couplage IPFS/RozoFS se comporte. Quel est le surcoût apporté par RozoFS, notamment en écriture, et quel est le gain que nous pouvons observer du fait de ne plus accéder à la table de hachage distribuée pour les objets stockés localement.

Temps d'accès

Temps moyen d'**écriture** (secondes)				Temps moyen de **lecture** (secondes)			
Nombre \ Taille	256 Ko	1 Mo	10 Mo	Nombre \ Taille	256 Ko	1 Mo	10 Mo
1	1,72	2,14	3,07	1	1,47	1,88	3,04
10	1,53	2,00	7,97	10	1,35	1,77	5,22
100	2,29	5,55	27,58	100	1,57	2,62	11,24

(a) – En utilisant une infrastructure de Cloud centralisé pour stocker tous les objets.

Temps moyen d'**écriture** (secondes)				Temps moyen de **lecture** (secondes)			
Nombre \ Taille	256 Ko	1 Mo	10 Mo	Nombre \ Taille	256 Ko	1 Mo	10 Mo
1	0,17	0,22	0,34	1	0,25	0,28	0,54
10	0,17	0,21	0,40	10	0,26	0,27	0,54
100	0,33	1,07	3,92	100	0,29	0,50	1,98

(b) – En utilisant l'approche par défaut d'IPFS.

Temps moyen d'**écriture** (secondes)				Temps moyen de **lecture** (secondes)			
Nombre \ Taille	256 Ko	1 Mo	10 Mo	Nombre \ Taille	256 Ko	1 Mo	10 Mo
1	0,18	0,23	0,38	1	0,14	0,18	0,31
10	0,17	0,22	0,43	10	0,14	0,18	0,36
100	0,33	1,08	3,97	100	0,19	0,36	1,83

(c) – En utilisant IPFS au-dessus d'une grappe RozoFS déployée sur chaque site.

TABLE 6.1 – Temps d'accès moyens (en secondes) pour lire ou écrire un objet dans différentes conditions de charge pour 3 sites.

Le Tableau 6.1 montre les temps d'accès moyen pour écrire ou lire un objet sur chacune des infrastructures composées de 3 sites. Le nombre d'objets indiqué correspond au nombre d'objets accédés en parallèle sur chaque site (donc 10 correspond à 10 objets écrits ou lus sur chacun des 3 sites, soit 30 accès au total).

Le Tableau 6.1(a) montre les temps d'accès pour l'infrastructure de Cloud. Dans ce scénario, chaque client écrit les objets sur une grappe IPFS accessible en 200 ms plutôt que sur leur site le plus proche.

Les résultats montrent des temps qui augmentent en fonction de la taille des objets. Il faut $1,72$ secondes en moyenne pour créer un simple objet de 256 Ko tandis que créer ce même objet, lorsqu'il fait 10 Mo, nécessite $3,07$ secondes. Le temps de transfert au travers de liens à forte latence devient important. Si nous augmentons le nombre d'objets accédés en parallèle, nous observons que le débit maximum qui peut être atteint en écriture est autour de 36 Mo/sec ($100 \times 10/27,58$). Pour les petits objets, la situation est pire puisque nous n'écrivons pas assez de données pour être capable d'utiliser toute la capacité du lien réseau ($0,256 \times 100/2,29 = 11,18$ Mbps soit $1,39$ Mo/s). En ce qui concerne les temps d'accès, ceux-ci sont meilleurs pour plusieurs raisons. Premièrement, la DHT n'est pas utilisée pour l'accès au objets stockés localement (*i.e.*, la DHT n'est pas appelée lorsque le nœud recevant la requête stocke un réplica de l'objet demandé). Deuxièmement, le débit TCP est meilleur en téléchargement qu'en émission des données, avec un maximum de 90 Mo/sec ($100 \times 10/11,24$). Cela est dû à la gestion des connexions TCP par Linux et à la granularité des envois. Le serveur IPFS envoie les données par gros paquets tandis que le client IPFS les envoie en plusieurs petits. Ces valeurs confirment l'impact de la latence sur le débit TCP comme rappelé précédemment.

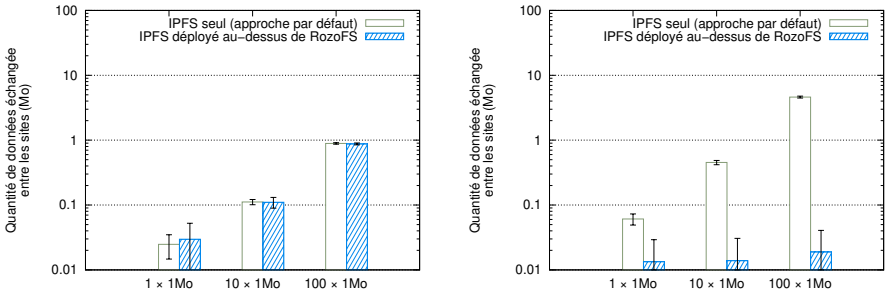
Le Tableau 6.1(b) présente les temps d'accès obtenus lorsqu'IPFS est déployé dans un environnement de Fog Computing. Les résultats montrent clairement le bénéfice de déployer un système de stockage objet à l'Edge. Les temps d'accès sont bien meilleurs que précédemment avec un débit maximal d'environ 500 Mo/sec (4 Gbps), ce qui représente la moitié de la performance maximale pour un lien de 10 Gbps. Deuxièmement, les performances sont réduites à cause des liens à forte latence entre les nœuds qui composent la DHT, ce qui n'est pas le cas dans l'approche utilisant une grappe IPFS dans le Cloud (latence de 0.5 ms entre les nœuds).

Le Tableau 6.1(c) montre les temps d'accès lorsqu'IPFS est déployé au-dessus de RozoFS. Nous observons tout d'abord, que dans les deux déploiements en mode Fog, les temps d'écriture sont similaires. Il faut $3,92$ secondes par objet pour écrire 100 objets de 10 Mo lorsqu'IPFS est déployé seul et $3,97$ seconds lorsqu'IPFS est couplé avec RozoFS. En d'autres mots, utiliser un *Scale-Out NAS* comme RozoFS ne dégrade pas les performances. Cela est surprenant car nous pouvons penser à tort que la complexité apportée par la couche RozoFS allait pénaliser les temps d'accès. Pour les lectures, nous observons que le couple IPFS/RozoFS permet d'améliorer légèrement les temps d'accès. Ceux-ci sont 34 % plus rapide en moyenne sur l'ensemble des valeurs du tableau par rapport à l'approche par défaut. Cela est particulièrement visible pour les petits objets. Par exemple il faut $0,14$ seconde pour lire 10 Mo tandis qu'il faut $0,25$ seconde lorsqu'IPFS est déployé seul (environ le double). Comme nous l'avons précédemment

décrit dans la Figure 5.2(b) de la Section 5.1.2, l'utilisation d'un *Scale-Out NAS* permet à IPFS de retourner directement un objet stocké sur le site, sans consulter la table de hachage distribuée. Le gain devient négligeable pour de gros objets car le temps d'accès à la DHT devient petit devant le temps nécessaire pour transférer l'objet au client. De plus, dans l'expérimentation correspondant au Tableau 6.1(b), le client IPFS a une chance sur 4 de contacter directement le nœud stockant l'objet désiré. En effet, le client sélectionne le serveur auquel il transmet sa requête de façon aléatoire et chaque site de Fog comporte 4 serveurs de stockage. Cette probabilité devient moins importante lorsque les sites utilisés comportent plus de nœuds de stockage, et le gain d'utilisation d'un *Scale-Out NAS* devient alors plus significatif.

Trafic réseau

Les Figures 6.1(a) et 6.1(b) montre la quantité de trafic réseau échangé entre les sites pendant les phases d'écriture et de lecture. Nous ne présentons pas les résultats lorsqu'IPFS est déployé en mode Cloud puisque dans ce cas, le trafic échangé entre les sites correspond à la quantité de données écrites ou lues.



(a) – Écriture (b) – Lecture

FIGURE 6.1 – Quantité moyenne cumulée de trafic échangé entre l'ensemble des sites pendant que les clients écrivent et lisent des objets sur leur site local.

Nous observons que la quantité de trafic émis entre les sites ne dépend que du nombre d'objets accédés et non de leurs tailles. Pour une meilleure lisibilité, nous ne présentons les résultats que pour des objets de 1 MB. La Figure 6.1(a) montre la quantité de trafic échangé lors de l'écriture. Sans surprise, cette quantité est identique pour les deux approches, puisque la localisation des objets est mise à jour de façon asynchrone dans la table de hachage distribuée.

Pour la lecture, le trafic généré par l'approche par défaut montre que plus il y a d'objets manipulés, plus la quantité de trafic échangé est élevée. Dans notre approche utilisant RozoFS, nous observons seulement le trafic envoyé à intervalles réguliers, permettant de maintenir la table de hachage distribuée (environ 13 Ko de données échangées entre les sites pour les trois cas présentés). Ainsi le trafic observé ne dépend que du temps mis pour réaliser l'expérimentation.

Temps moyen de lecture (secondes) **Première lecture**				Temps moyen de lecture (secondes) **Seconde lecture**			
Nombre \ Taille	256 Ko	1 Mo	10 Mo	Nombre \ Taille	256 Ko	1 Mo	10 Mo
1	1,39	1,92	13,07	1	1,01	1,85	3,70
10	1,01	1,92	6,48	10	0,70	1,31	5,95
100	0,94	2,02	9,76	100	0,71	1,37	6,08

(a) – En utilisant IPFS seul.

Temps moyen de lecture (secondes) **Première lecture**				Temps moyen de lecture (secondes) **Seconde lecture**			
Nombre \ Taille	256 Ko	1 Mo	10 Mo	Nombre \ Taille	256 Ko	1 Mo	10 Mo
1	1,35	3,86	13,21	1	0,15	0,19	0,31
10	1,11	2,17	8,40	10	0,14	0,19	0,35
100	1,09	2,51	9,22	100	0,33	0,46	1,86

(b) – En utilisant IPFS déployé au dessus d'une grappe RozoFS locale à chaque site.

TABLE 6.2 – Temps moyens (en secondes) pour lire un objet deux fois avec IPFS en utilisant l'approche par défaut (a) et en utilisant notre couplage (b) dans différentes conditions de charge pour 3 sites.

Pour conclure, l'ajout d'un *Scale-Out NAS* à IPFS ne dégrade pas les performances en écriture. Les temps d'accès observés sont similaires, avec ou sans RozoFS. En lecture, en revanche, l'utilisation d'un Scale-Out NAS permet de ne pas consulter la table de hachage distribuée lorsque les objets demandés sont stockés localement sur le site. Cela permet d'améliorer les performances, notamment lors de l'utilisation de petits objets, pour lesquels cet accès à la DHT représentait une part importante du temps d'accès. Dans le même temps, cette non-utilisation de la table de hachage distribuée élimine le trafic réseau inter-sites lors de la lecture.

6.1.3 Lecture depuis un site distant

Dans cette section, nous discutons des résultats obtenus dans un scénario d'accès distant, c'est-à-dire, lorsqu'un client écrit un objet sur son site et que celui-ci est accédé (à deux reprises) par un autre client, situé sur un autre site. Seuls les résultats en lecture sont présentés car les résultats en écriture sont identiques à ceux que nous venons de présenter. Nous n'évaluons pas non plus les performances d'une infrastructure de type Cloud puisqu'il n'y aura pas de changement par rapport au scénario précédent : les clients étant toujours à la même distance de l'infrastructure.

Temps d'accès

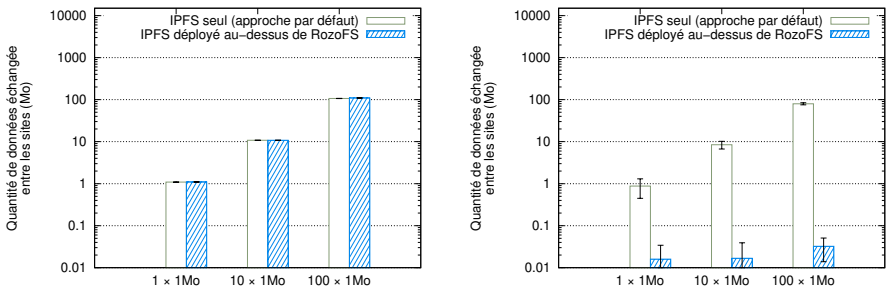
Les Tableaux 6.2(a) et 6.2(b) montrent les temps d'accès pour les deux lectures successives. Lors de la première lecture, les temps d'accès des deux approches sont du même ordre de grandeur. En effet, pour les deux approches, la donnée demandée n'est pas sur le site local.

La table de hachage distribuée doit être contactée puis l'objet téléchargé. Cela correspond à ce que nous décrivons dans les diagrammes de séquences, sur les Figures 5.0(c) et 5.2(c) des Sections 5.1.1 et 5.1.2. Les échanges réseau sont les mêmes : le nœud interrogé par le client consulte la DHT et télécharge le réplica stocké sur le premier site. La seule différence est le nœud distant qui, dans notre approche, ira chercher l'objet sur le système de fichiers distribué sur son site.

Pour la seconde lecture, nous observons en revanche de meilleurs temps d'accès avec notre approche. Par exemple avec RozoFS, lorsque 100 objets de 10 Mo sont lus en parallèle, il faut 1, 86 seconde par objet tandis qu'il en faut 6, 08 secondes lorsqu'IPFS est utilisé seul. Bien que de nouveaux réplicas soient créés localement dans les deux approches, la probabilité d'interroger lors de la seconde lecture, le même nœud que lors de la première lecture n'est que de 0, 25 car chaque site comporte 4 serveurs de stockage et que le nœud sollicité est choisi aléatoirement. Dans l'approche par défaut, lorsque le client n'interroge pas le même nœud, le processus d'accès est alors le même que lors de la première lecture. La seule différence mineure est qu'une fois la DHT interrogée, l'objet peut être téléchargé soit du site distant, soit du site local, par le nœud sur lequel un nouveau réplica de l'objet a été créé lors de la première lecture.

Dans l'approche couplant IPFS à RozoFS, le système de fichiers distribué et partagé entre tous les nœuds du site fait que la DHT n'est jamais consultée lors de la seconde lecture.

Trafic réseau



(a) – Première lecture (b) – Seconde lecture

FIGURE 6.2 – Quantité moyenne cumulée de trafic échangé entre l'ensemble des sites pendant que les clients effectuent deux lectures successives.

Les Figures 6.2(a) et 6.2(b) montrent la quantité de trafic réseau échangé entre les sites pendant les deux lectures. Pour la première lecture, la même quantité est échangée dans les deux approches puisque comme expliqué précédemment, dans les deux approches l'objet doit être localisé puis téléchargé du site où il a été créé. Ce trafic inter-sites est dominé par l'échange de données plus que par l'utilisation de la table de hachage distribuée. Le trafic de la DHT n'est pas significatif au regard du trafic généré par le

transfert des objets (4 % environ).

Pour la seconde lecture, nous observons une plus grande quantité de trafic lorsqu'IPFS est utilisé seul. En effet, dans cette approche, l'objet est localisé avec la DHT dans $\frac{3}{4}$ des cas, et lorsque l'objet est téléchargé, tous les réplicas sont sollicités : le réplica créé localement lors de la première lecture mais aussi le réplica original situé sur le site distant. Dans notre approche, la requête est directement satisfaite par le nœud IPFS interrogé grâce au système de fichiers distribué sous-jacent.

6.1.4 Conclusion

Les résultats principaux de cette expérience sont :

- Le *Scale-Out NAS* n'a pas d'impact négatif sur les performances, même si 50 % de données supplémentaires sont stockées pour la tolérance aux pannes ;
- Les lectures locales sont satisfaites par le système de fichiers distribué sous-jacent, évitant d'accéder à la DHT. Cela aboutit à une amélioration moyenne de 34 % des temps d'accès en lecture par rapport à un déploiement d'IPFS dans sa configuration par défaut et à la quasi-suppression des trafics réseaux inter-sites ;
- Lors d'un accès distant, le couplage n'a pas d'impact sur les temps d'accès et le trafic réseau inter-sites. En revanche, une fois l'objet répliqué, les mêmes améliorations que lors d'un accès local sont observées.

6.2 Évaluation de l'approche en arbre

Dans cette partie, nous évaluons notre gestion des localisations en arbre. Afin de comprendre le comportement de notre approche, nous proposons de l'expérimenter sur des micro-comparaisons avant de réaliser une évaluation de plus grande envergure.

Ce travail a été présenté à RESCOM 2018 [Confais et al. 2018c] avant d'être soumis dans la conférence IEEE Globecom 2018 [Confais et al. 2018a]

6.2.1 Matériels & méthodes

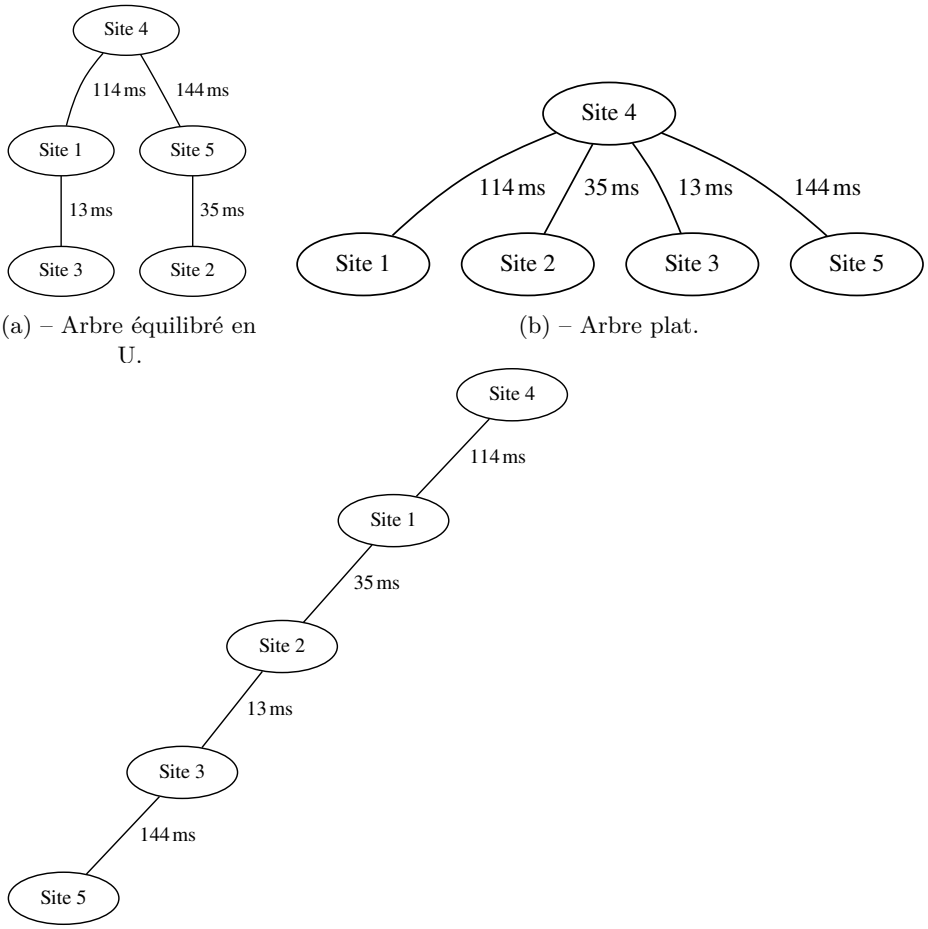
Dans cette section, nous émulons chaque site de Fog par seulement un nœud IPFS. Chaque nœud IPFS héberge également un serveur DNS. Afin de réduire le travail de développement, nous implémentons notre approche en stockant les enregistrements de localisation au sein de serveurs DNS. Nous modifions le mécanisme de routage d'IPFS afin d'interroger les serveurs DNS plutôt qu'une DHT. Les mises à jour des serveurs DNS est réalisée grâce au protocole *Dynamic DNS protocol* (DDNS).

Tous les serveurs sont responsables de la même zone, mais stockent des enregistrements différents. Afin d'implémenter notre stratégie de routage dans IPFS, nous utilisons la bibliothèque Go DNS[1].

[1] https://github.com/miekg/dns

Enfin, pour émettre les requêtes du bas de l'arbre DNS vers le haut, nous n'envoyons pas les requêtes à un résolveur DNS traditionnel mais nous interrogeons directement les serveurs faisant autorité. Lorsque le service IPFS démarre, il effectue une requête à partir de son propre nom de site. Cette requête est effectuée de la racine vers le site en question, ce qui permet au nœud IPFS de collecter l'ensemble des sites sur le chemin entre lui et la racine. De cette façon, il est ensuite possible d'émettre les requêtes en utilisant une approche *bottom-up*, c'est-à-dire en interrogeant les serveurs depuis la position actuelle et en remontant vers la racine de l'arbre.

Les noms des objets sont suffixés par le nom d'un site, nous avons supprimé le mécanisme d'IPFS permettant de générer un nom d'objet à partir de son contenu. Pour une comparaison équitable, nous avons effectué cette modification, non seulement dans notre version mais aussi dans la version standard utilisant une DHT[2].



(a) – Arbre équilibré en U.

(b) – Arbre plat.

(c) – Arbre profond.

FIGURE 6.3 – Topologies utilisées pour les micro comparaisons.

[2] https://github.com/bconfais/go-ipfs/tree/dht_name_based

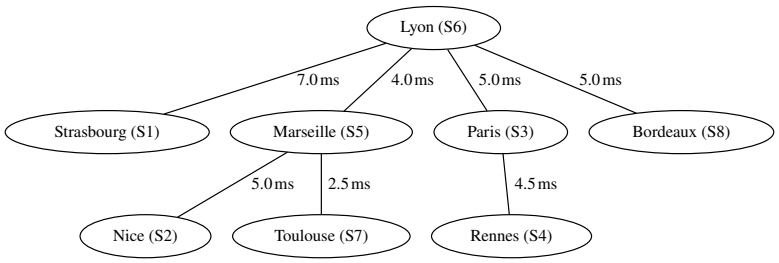


FIGURE 6.4 – Arbre généré avec notre fonction de coût $c = 1, 2$

L'implémentation de notre protocole a nécessité d'écrire environ 500 lignes de code en Go. Le code source est disponible à l'adresse suivante : https://github.com/bconfais/go-ipfs/tree/dns.

Nous utilisons respectivement les topologies de la Figure 6.3 et de la Figure 6.4 pour la micro et la macro-comparaison.

Pour chacune des 4 topologies, les objets sont écrits sur le premier site (Site 1) et sont ensuite accédés en parallèle depuis les autres sites. Le choix du Site 1 comme origine pour réaliser les écritures est purement arbitraire. Pour les lectures, chaque objet est accédé depuis un site qui n'a pas accédé à l'objet auparavant. De cette façon, nous ne lisons jamais d'objets stockés localement pour lesquels, le processus de localisation n'est pas nécessaire. Le Site 1 n'effectue aucune lecture puisqu'il s'agit du site sur lequel les objets sont écrits. Il sera capable de répondre directement à toute requête de lecture émise vers celui-ci sans avoir besoin de localiser l'objet demandé.

Nous mesurons le temps de localisation de chaque objet mais aussi le nombre de liens réseaux traversés (*i.e.*, le nombre de sauts). Nous ne considérons pas le temps d'accès aux données car une gestion différente de la localisation peut mener à ne pas accéder aux mêmes réplicas de données. Par exemple, chaque clé de la table de hachage distribuée est associée à la localisation de tous les réplicas ce qui n'est pas forcément le cas dans notre approche. En effet, dans notre approche, comme présenté dans la Section 5.2.4, tous les serveurs de localisation ne sont pas mis à jour à chaque création de réplica. Cela peut aboutir au fait que les mêmes réplicas de données ne sont pas accédés avec les deux approches. Nous notons « DHT kX », la table de hachage distribuée dans laquelle chaque enregistrement est répliqué X fois.

Comme précédemment, les tests sont effectués sur la plateforme Grid'5000 et les latences sont émulées avec *Linux Traffic Control Utility* (`tc`). Nous utilisons 1 000 objets de 4 Ko chacun. Comme nous mesurons uniquement le temps de localisation des objets, leur taille ne nous importe pas et n'influe pas sur nos résultats. Nous précisons également que nous ne cumulons pas les deux approches, c'est-à-dire que nous utilisons IPFS seul et non couplé avec le *Scale-Out NAS* Comme pour notre précédente évalusation, nous avons écrit un script python qui déploie IPFS et exécute l'expérimentation. Ce script est disponible à l'adresse suivante : https://github.com/bconfais/benchmark/tree/master/dns.

Toutes les expérimentations sont exécutées 10 fois. Les moyennes sont calculées sur

95 % des valeurs (les valeurs extrêmes sont supprimées) et les écarts-types ne sont pas représentés mais ont des valeurs autour de 0.1 s.

6.2.2 Micro-comparaisons

Dans cette section, nous effectuons des micro-comparaisons à partir des topologies de la Figure 6.3. L'objectif est de valider notre plan expérimental. Ces topologies sont construites à partir de 5 sites de la matrice de latence donnée sur le site web de *Wondernetwork*[3].

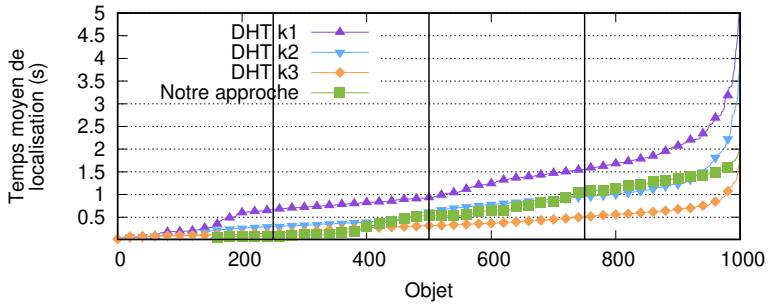
Première topologie : arbre équilibré

Nous commençons par évaluer notre approche à partir de l'arbre équilibré de la Figure 6.3(a). La Figure 6.5(a) montre le temps pour localiser les objets, à la fois pour la DHT et pour notre approche, lorsque les objets sont accédés pour la première fois. Les objets sont triés par leur temps de localisation. Il apparaît que localiser un objet avec notre protocole nécessite dans le pire des cas 1, 98 s (temps pour le dernier objet), ce qui est plus rapide que la DHT avec un seul réplica, où 4, 79 s sont nécessaires. En revanche, la DHT utilisant 3 réplicas nécessite seulement 1, 740 s. Nous précisons quand même que lors de la première lecture, notre approche n'utilise que deux réplicas par rapport à la DHT. Comparer notre approche avec une DHT à 3 réplicas n'est donc pas équitable lors d'une première lecture.

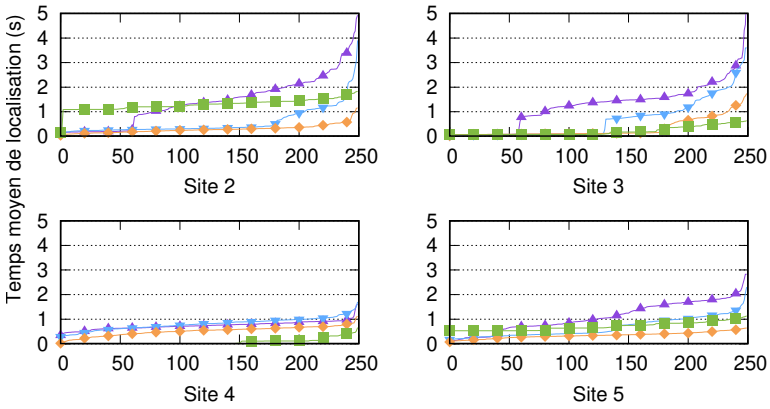
Les lignes verticales montrent les groupes d'objets théoriques pour lesquels la localisation devrait être atteinte avec la même latence. Lors de la première lecture, tous les objets sont localisés, soit en contactant le Site 1, soit en contactant le Site 4. Par exemple, le premier groupe d'objets montre les objets lus depuis le Site 4 pour lesquels la localisation est stockée localement. Le second groupe est composé principalement d'objets lus depuis le Site 3, pour lesquels la localisation est trouvée sur le Site 1, atteignable en 13 ms. Enfin, les objets 500 à 750 montrent les objets lus depuis le Site 5 et le dernier seuil délimite les objets lus depuis le Site 2. Ces deux sites accèdent au Site 4. Les non-linéarités que nous observons près des seuils montrent que le résultat obtenu correspond à ce à quoi nous nous attendons. Parce que les objets sont triés par temps de localisation, les seuils ne délimitent pas exactement ce qui se passe site par site. Pour mieux comprendre les comportements sur chaque site, nous séparons les objets en fonction du site sur lequel ils sont accédés. La Figure 6.5(b) montre que les temps de localisation des objets sont différents selon les sites et selon la latence réseau pour joindre les autres sites à partir de celui-ci. Cela est vrai à la fois pour notre approche mais aussi lorsqu'une table de hachage distribuée est utilisée. Par exemple, d'après la Figure 6.3(a), le Site 4 ne peut pas atteindre un autre site en moins de 114 ms.

De même, dans notre approche, lors de la première lecture, chaque site localise tous les objets en trouvant les enregistrements de localisation au même endroit, ce qui n'est pas le cas dans la DHT où les enregistrements sont répartis uniformément sur l'ensemble des nœuds. C'est pourquoi, sur la Figure 6.5(b) nous n'observons pas de non-linéarité

[3] https://wondernetwork.com/pings



(a) – Tous les sites



(b) – Par site

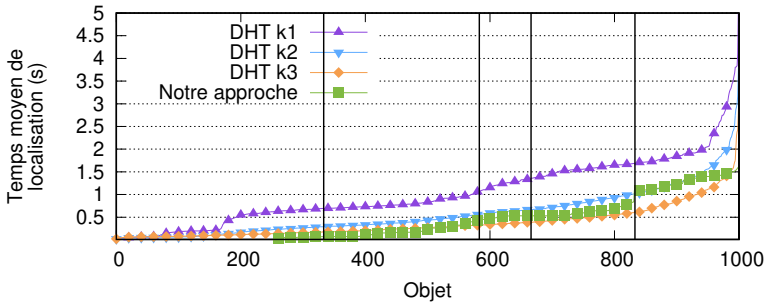
FIGURE 6.5 – Temps moyens pour localiser chaque objet pour la première lecture, (a) en considérant l'ensemble des objets ou (b) en détaillant les accès site par site. Les objets sont triés par temps croissants de localisation.

dans notre approche pour un site donné. Pour donner un exemple, le Site 2 localise tous les objets en atteignant le Site 4, le Site 3 en contactant le Site 1, etc.

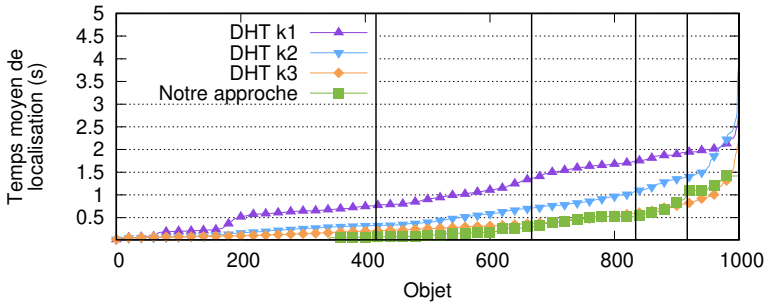
Nous signalons également que les derniers objets, lus avec des temps toujours croissants sont dûs à un mauvais parallélisme d'IPFS que nous avons validé par des accès séquentiels. Lorsque les 1 000 objets sont demandés de manière séquentielle, tous les objets demandés depuis un même site sont localisés avec un temps similaire, montrant des paliers très nets, comme nous pouvons le voir dans la Figure 6.7.

Ce mauvais parallélisme explique aussi pourquoi les temps observés sont plus élevés que les temps que nous calculons depuis l'arbre. En effet, dans l'arbre, le temps d'accès le pire est pour le Site 2 qui atteint la racine en $35 \times 2 + 144 = 214$ ms, ce qui signifie que la localisation doit être trouvée, dans le pire des cas en 428 ms (la latence doit être multipliée par 2 pour prendre en compte les messages de réponse). Pourtant, dans notre évaluation, jusqu'à $1,98$ s sont nécessaires au Site 2 pour localiser un objet. Pour supprimer ce biais, nous évaluons les performances en nombre de sauts plutôt qu'en temps absolus.

La Figure 6.6(a) montre que pour la seconde lecture, le temps de localisation ne varie



(a) – Seconde lecture



(b) – Troisième lecture

FIGURE 6.6 – Temps moyens pour localiser chaque objet pour la seconde lecture (a) et la troisième lecture (b). Les objets sont triés par temps croissants de localisation. Les lignes verticales montrent les seuils théoriques que nous devons observer dans notre approche.

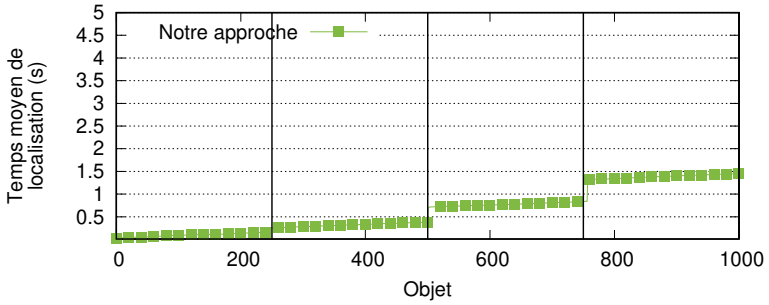
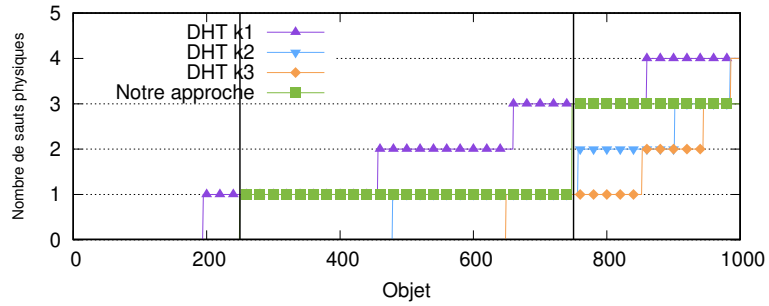
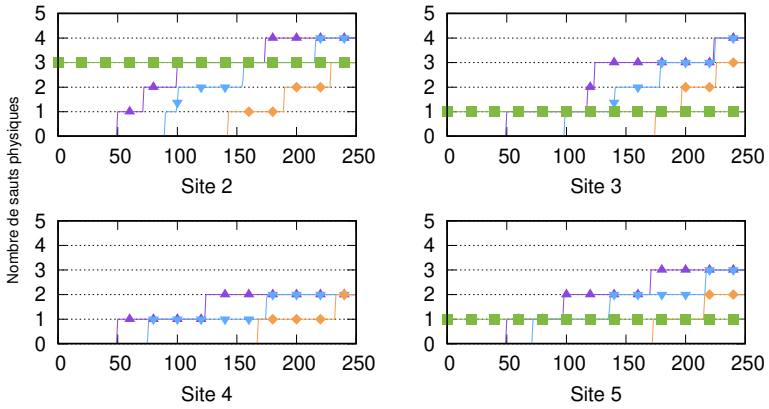


FIGURE 6.7 – Temps pour localiser chaque objet pour la première lecture, lorsque les objets sont accédés de manière séquentielle.

pas avec la DHT mais se réduit avec notre approche qui réplique l'enregistrement de localisation au fur et à mesure des accès. Pour la troisième lecture (Figure 6.6(b)), notre approche devient meilleure que la DHT utilisant trois réplicas car la localisation est placée proche des sites où elle est utilisée au lieu d'être répartie de façon uniforme. Notre approche nécessite 1,42 s pour localiser les 1 000 objets tandis que la DHT a besoin de 2, 21 s.



(a) – Tous les sites



(b) – Par site

FIGURE 6.8 – Nombre moyen de sauts physiques pour localiser chaque objet dans l'arbre équilibré lors de la première lecture. Les objets sont triés par temps croissants de localisation.

Comme nous créons de nouveaux réplicas de localisation au fil des lectures, les seuils théoriques varient. Ainsi, pour la seconde lecture, la localisation sera trouvée localement pour 333 objets car le Site 5 stocke maintenant la localisation des objets lus depuis le Site 2 à la première lecture. Une observation similaire est faite pour le Site 2 qui peut maintenant accéder à certaines localisations en contactant le Site 5 plutôt que de remonter jusqu'à la racine.

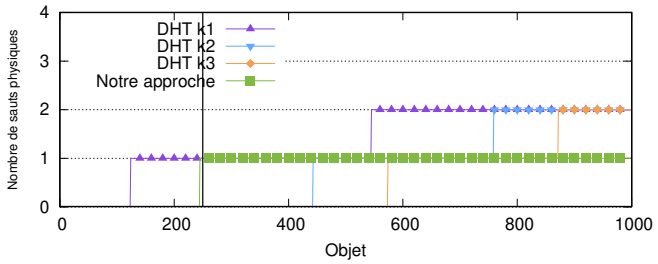
La Figure 6.8(a) montre le nombre de sauts physiques pour localiser chaque objet. À cause de notre protocole itératif, le nombre de sauts peut seulement être de 0, 1 ou 3. En effet, le Site 2 contacte d'abord le Site 5 en un saut puis le Site 4 en traversant deux liens physiques. Nous notons que les latences réseau n'ont pas d'impact sur les nombres de sauts, ceux-ci dépendent uniquement de la topologie de l'arbre.

Les seuils théoriques sont les mêmes que pour les temps d'accès mais ici, les objets localisés en un saut, que ce soit du Site 3 ou du Site 2 sont rassemblés dans le même groupe.

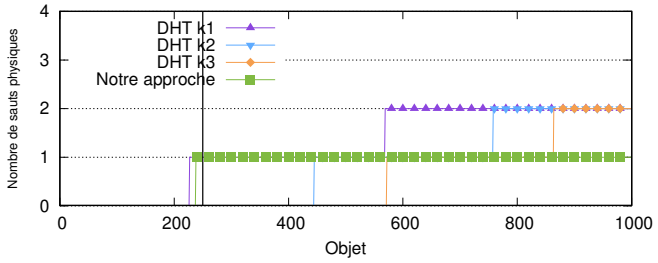
La conclusion de cette expérimentation est qu'interroger les nœuds proches en premier

et créer de nouveaux réplicas des enregistrements de localisation permet d'obtenir de meilleures performances que l'utilisation d'une table de hachage distribuée. Créer des enregistrements de localisation à la volée, au fur et à mesure des lectures permet de réduire le nombre de sauts ainsi que le temps de localisation. Contrairement à la DHT où le temps de localisation dépend uniquement de la latence qui connecte le site où l'objet est demandé aux autres sites, dans notre approche, le temps de localisation dépend également de la popularité de l'objet. Plus un objet possède de réplicas, plus son temps de localisation sera faible.

Seconde topologie : arbre plat



(a) – Première lecture



(b) – Quatrième lecture

FIGURE 6.9 – Nombre moyen de sauts physiques pour localiser chaque objet avec l'arbre plat lors de la (a) première et (b) quatrième lecture. Les objets sont triés par temps croissants de localisation.

La seconde topologie que nous évaluons est présentée dans la Figure 6.3(b). Dans cette topologie, notre approche ne peut pas bénéficier des nouveaux réplicas des enregistrements de localisation. Lorsque la localisation n'est pas trouvée localement, le nœud racine est directement contacté.

Néanmoins, nous montrons que même dans ce scénario, notre approche donne de meilleures performances que l'utilisation d'une table de hachage distribuée. Nous nous concentrons sur le nombre de sauts. L'utilisation des latences n'apporterait en plus, que de distinguer clairement les accès effectués depuis le Site 5, site connecté à la racine par un lien réseau élevé.

La Figure 6.9 montre que le nombre de sauts n'est pas réduit entre la première et la quatrième lecture, que ce soit avec notre approche ou en utilisant une DHT. Dans

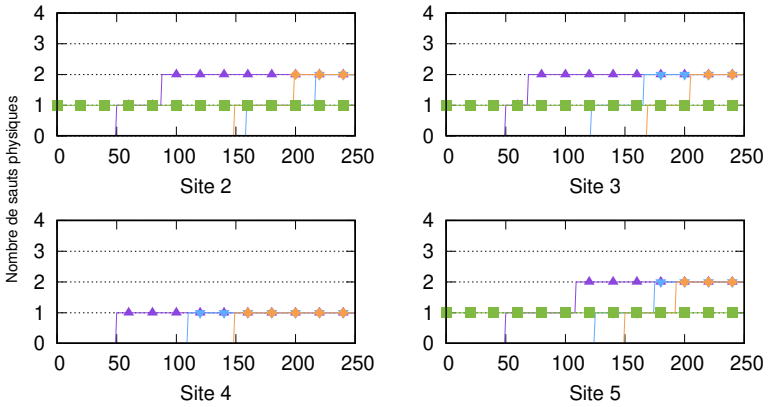


FIGURE 6.10 – Nombre moyen de sauts physiques pour localiser chaque objet depuis chaque site avec l'arbre plat lors de la première lecture.

notre approche, la localisation est toujours trouvée sur le Site 4. Nous observons une non-linéarité, délimitant les objets lus depuis le Site 4 pour lesquels la localisation est trouvée localement et les autres sites qui ont besoin d'effectuer un saut.

Si nous regardons le nombre de sauts dans la Figure 6.10, nous observons que notre approche réalise moins de sauts que la DHT car dans la DHT, la localisation peut être trouvée sur un nœud frère, nécessitant l'utilisation de deux liens réseaux (du site source vers le Site 4 puis du Site 4 vers le site de destination).

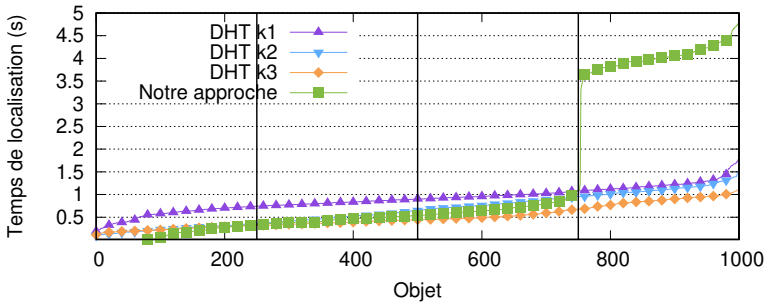
La conclusion de cette expérimentation est que même dans une situation où notre approche ne bénéficie pas des nouveaux réplicas, le trafic reste mieux confiné que dans une table de hachage distribué. Nous notons cependant que cela peut surcharger le nœud racine. Cela met également en avant le besoin d'avoir un arbre équilibré.

Troisième topologie : arbre profond

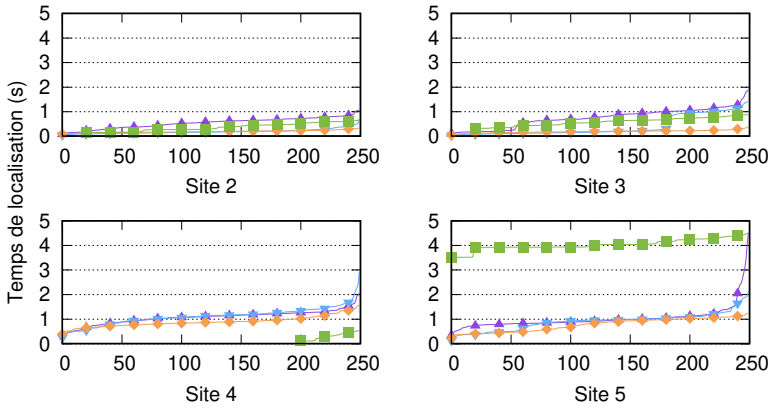
La dernière micro-topologie que nous évaluons est l'autre cas extrême dans lequel tous les sites sont organisés dans un arbre dégénéré, comme cela est montré dans la Figure 6.3(c).

Nous montrons qu'avoir des liens à forte latence proche des feuilles impacte de façon très négative notre approche, particulièrement lorsque de nombreux sauts sont nécessaires pour localiser un objet.

Nous observons dans la Figure 6.12(a) que lire les 1 000 objets dans notre approche, en 4,5 s est pire qu'utiliser une table de hachage distribuée qui ne nécessite que 1,7 s. La Figure 6.11(b) montre que ce résultat est dû au Site 5, connecté aux autres sites par un lien à forte latence (144 ms). Dans la DHT, ce site trouve la localisation en un seul saut logique, par conséquent le nœud stockant la localisation est atteint directement et le lien à forte latence n'est traversé qu'une seule fois. Dans notre approche, le lien à forte latence est utilisé une première fois pour joindre le Site 3, puis une seconde fois pour atteindre le Site 2 et enfin, une troisième fois pour atteindre le Site 1. Cela explique pourquoi les temps de localisation sont plus élevés dans notre approche. Comme le lien a



(a) – Tous les sites



(b) – Par site

FIGURE 6.11 – Temps moyens pour localiser chaque objet avec l'arbre profond lors de la première lecture, (a) en considérant l'ensemble des objets ou (b) en détaillant les accès site par site.

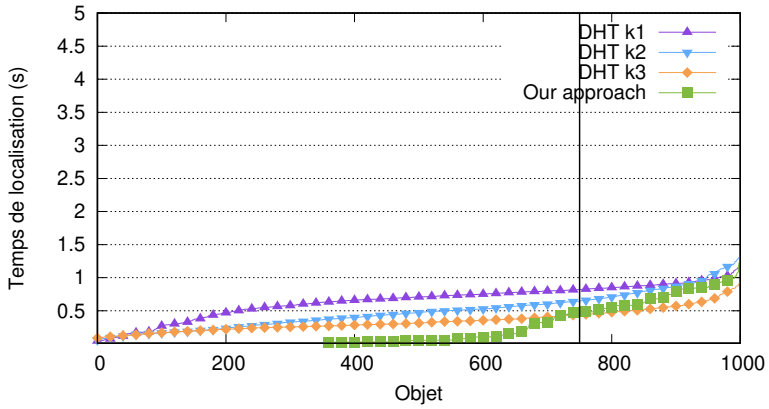
forte latence est plus sollicité, le temps de localisation est plus élevé. Cela nous rappelle que nous devons générer un arbre dans lequel les liens situés près des feuilles doivent être ceux avec la latence les plus faibles.

La Figure 6.12 montre que les temps sont réduits dans la dernière lecture car le Site 5 est alors capable de trouver toutes les localisations souhaitées sur le Site 3, accessible en un seul saut.

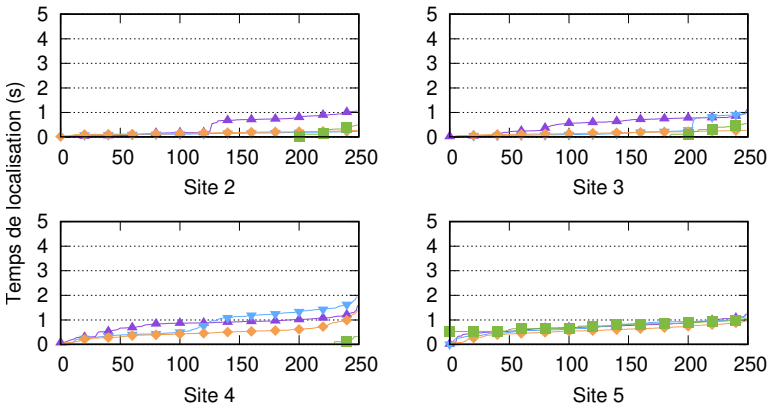
La conclusion de cette expérience est que les liens à forte latence situés dans les profondeurs de l'arbre dégradent significativement les temps de localisation puisque plus un lien est profond dans l'arbre, plus celui-ci sera sollicité pour atteindre la racine de l'arbre (sollicité n fois où n est la profondeur de l'arbre). En revanche cette forte utilisation sera compensée lors de la création de nouveaux réplicas de localisation. Une fois des réplicas créés dans les profondeurs, atteindre la racine ne sera plus nécessaire.

6.2.3 Macro-comparaison

Après avoir effectué des micro-comparaisons pour comprendre comment notre protocole se comporte, nous l'évaluons maintenant sur une topologie réelle.



(a) – Tous les sites

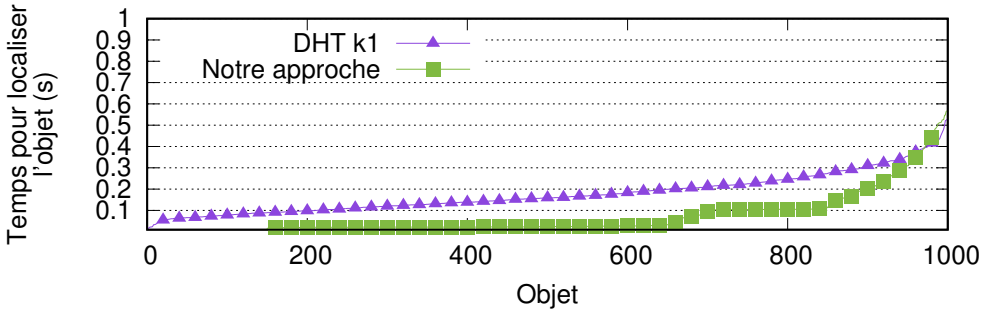


(b) – Par site

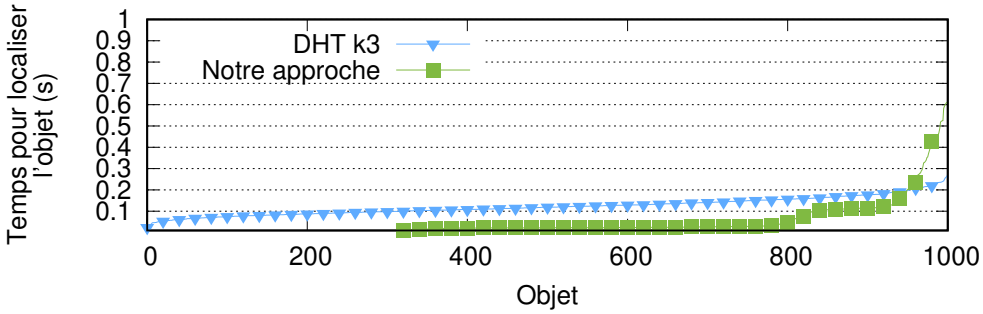
FIGURE 6.12 – Temps moyens pour localiser chaque objet avec l'arbre profond lors de la quatrième lecture, (a) en considérant l'ensemble des objets ou (b) en détaillant les accès site par site.

Nous considérons le sous-graphe du réseau RENATER, illustré dans la Figure 5.15. Afin d'évaluer notre approche, nous utilisons l'arbre présenté dans la Figure 5.16 qui a été calculé en utilisant l'approche présentée dans la Section 5.2.6. Pour l'approche utilisant la table de hachage distribuée, nous calculons les plus courts chemins (à l'aide de l'algorithme de Dijkstra) entre chaque couple de nœuds, de sorte que la localisation puisse être accédée avec la plus faible latence possible. Dit autrement, la table de hachage distribuée bénéficie d'un routage optimal, ce qui n'est pas le cas dans notre approche où les échanges suivent les branches de l'arbre couvrant.

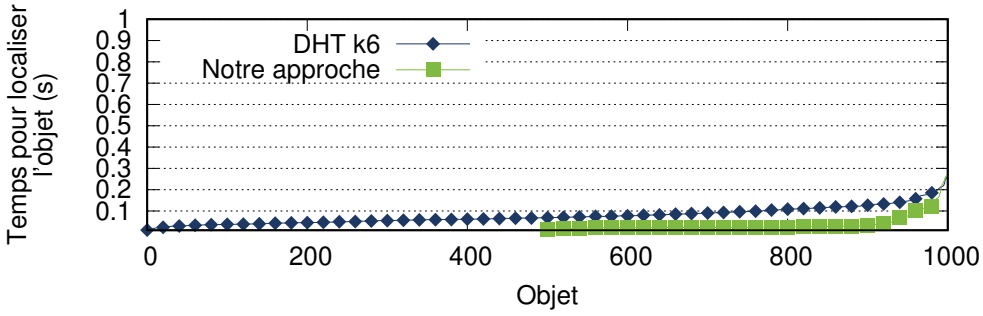
Nous exécutons le même scénario que dans la section précédente : 1 000 objets sont écrits sur le Site 1 (Strasbourg) et sont lus successivement depuis les autres sites. La Figure 6.13 montre les temps pour localiser les objets lors de la première, troisième et sixième lectures. Comme dans notre approche, des réplicas de localisation peuvent



(a) – Première lecture



(b) – Troisième lecture



(c) – Sixième lecture

FIGURE 6.13 – Temps moyens pour localiser les objets lors de la première (a), troisième (b) et sixième (c) lecture.

être créés sur tous les sites. Nous évaluons la table de hachage jusqu'à l'utilisation de 6 réplicas. En effet, lors de la première lecture, un seul réplica est utilisé : tous les sites accèdent à la localisation en interrogeant le site racine (Lyon).

Nous observons que pour l'ensemble des lectures (Figures 6.13(a), 6.13(b) et 6.13(c)), notre approche montre de meilleures performances que l'utilisation d'une table de hachage distribuée. Cela s'explique simplement parce que notre arbre respecte la topologie physique et que même dans le pire des cas, atteindre le nœud racine situé à équidistance des autres sites s'avérait souvent plus intéressant qu'une DHT qui pourrait nécessiter de traverser le réseau en entier. La Figure 6.13(a) montre les temps de localisation mesurés lors de la première lecture. Nous observons une non-linéarité autour de 600 objets. Ce seuil délimite d'un côté les sites pour lesquels la localisation est trouvée en un seul saut (Sites 5, 3 et 8) et les objets qui nécessitent 3 sauts physiques (Sites 2, 4 et 7)

La troisième lecture est également meilleure qu'une table de hachage distribuée contenant 3 réplicas, comme le montre la Figure 6.13(b). Nous devons toutefois signaler que comparer notre approche avec une DHT contenant trois réplicas pourrait être vu comme inéquitable puisque dans notre approche le nombre de réplicas de localisation dépend des endroits où les objets ont été lus précédemment. Par exemple, un objet lu à Lyon, puis à Marseille ne bénéficie que de deux réplicas lorsque la troisième lecture est exécutée alors qu'un objet lu à Nice puis à Rennes bénéficie de 5 réplicas de localisation lors de la troisième lecture. Nous pouvons toutefois remarquer que notre approche est meilleure que l'utilisation de la table de hachage distribuée même si les 50 derniers objets sont localisés avec un temps plus élevé car de nouveaux réplicas n'ont pas encore été créés. Enfin, la Figure 6.13(c) montre également de meilleures performances lors de la sixième lecture, lorsqu'un réplica de la localisation de l'objet se trouve sur l'ensemble des sites (sauf un si le dernier site qui accède à l'objet est une feuille de l'arbre).

Bien que les temps de localisation soient assez hétérogènes, nous pouvons mesurer en moyenne un gain de 20 % des temps de localisation par rapport à l'utilisation d'une table de hachage distribuée.

6.2.4 Conclusion

Ces expérimentations montrent qu'en limitant le nombre de sauts, en confinant et en réduisant le trafic inter-sites, notre approche permet aux nœuds d'accéder à l'enregistrement de localisation plus rapidement qu'en utilisant une table de hachage distribuée. Nous avons montré que même un arbre « plat » pouvait bénéficier d'une telle approche dans la mesure où le nœud racine a une position centrale dans le réseau. Cela limite la latence maximale puisque dans le pire des cas, les requêtes ne traversent que la moitié du réseau. Toutefois, il faut faire attention avec une telle topologie, de ne pas surcharger le nœud racine. Également, notre implémentation utilisant des requêtes transmises de façon itératives est désavantagée lors de l'utilisation d'arbres profonds mais la création dynamique de réplicas permet de compenser cet inconvénient. Enfin, nous avons également montré qu'un arbre construit à partir d'une topologie réelle permettait également de réduire les temps d'accès d'environ 20 %.

Nous devons cependant insister sur le fait que les tables de hachage distribuées ne

comportaient que peu de nœuds et que les localisations étaient atteintes en un seul saut logique. Nous pouvons donc penser que les performances de la table de hachage distribuée observées sont meilleures que ce que nous observerions dans un environnement réel.

6.3 Utilisation simultanée des plateformes Grid'5000 et FIT/IoT-Lab

Après avoir évalué expérimentalement sur la plateforme Grid'5000 nos propositions, nous cherchons à mettre en place une évaluation plus réaliste, d'une plateforme de Fog Computing, interconnectant l'Internet des Capteurs avec l'Internet des Serveurs. Pour ce faire, nous avons utilisé la plateforme de tests FIT/IoT-Lab qui fournit des capteurs connectés à l'Internet [Adjih et al. 2015]. Ce travail a été présenté lors de l'école de printemps FIT-G5K 2018 [Confais et al. 2018e]. Évaluer notre approche dans un tel environnement, permettrait non seulement d'utiliser des clients plus représentatifs d'un environnement de Fog Computing mais permettrait également de manipuler de véritables données, et d'avoir un véritable modèle de charge. À notre connaissance, il n'y a eu à ce jour qu'une autre étude abordant l'interconnexion de ces deux plateformes [Donassolo et al. 2018]. Les travaux présentés par la suite sont donc précurseurs.

6.3.1 Présentation de FIT/IoT-Lab

FIT-IoT-Lab est une plateforme d'expérimentation fournissant différents types de machines. Les machines fournies sont des capteurs connectés à l'Internet, et ayant peu de puissance de calcul et peu de mémoire. Des nœuds de type M3 fonctionnent avec un processeur de type ARM et possèdent un équipement radio respectant le protocole IEEE 802.15.4 (Zigbee), limité à 250 Kbps. Ils peuvent fonctionner avec différents systèmes d'exploitation comme FreeRTOS [Barry 2010], Contiki [Dunkels et al. 2011] ou comme RIOT [Baccelli et al. 2013]. Les nœuds de type A8, quant à eux, fournissent un environnement Linux et peuvent être associés à un nœud M3. Les nœuds M3 peuvent soit être interconnectés à l'aide de l'interface radio, soit avoir le rôle de routeur de bordure, permettant de faire le lien entre le réseau Ethernet câblé et le réseau radio. Certains nœuds peuvent être embarqués au sein de robots mobiles. La Figure 6.14 donne un aperçu général de la plateforme. Comme avec Grid'5000, les nœuds sont répartis sur différents sites (Grenoble, Lille, Lyon, Saclay, Strasbourg).

6.3.2 Proposition / expérimentation

L'expérience a consisté à coupler la plateforme Grid'5000 à la plateforme FIT/IoT-Lab. Tandis que la plateforme Grid'5000 fournit des nœuds avec de grandes capacités de calcul et de stockage, permettant d'émuler un site de Fog Computing, la plateforme FIT/IoT-Lab fournit des capteurs avec peu de ressources qui peuvent utiliser les ressources du site de Fog. Dit autrement, nous souhaitons que les nœuds de la plateforme FIT/IoT-Lab servent à émuler les clients des nœuds de stockage IPFS hébergés sur la plateforme

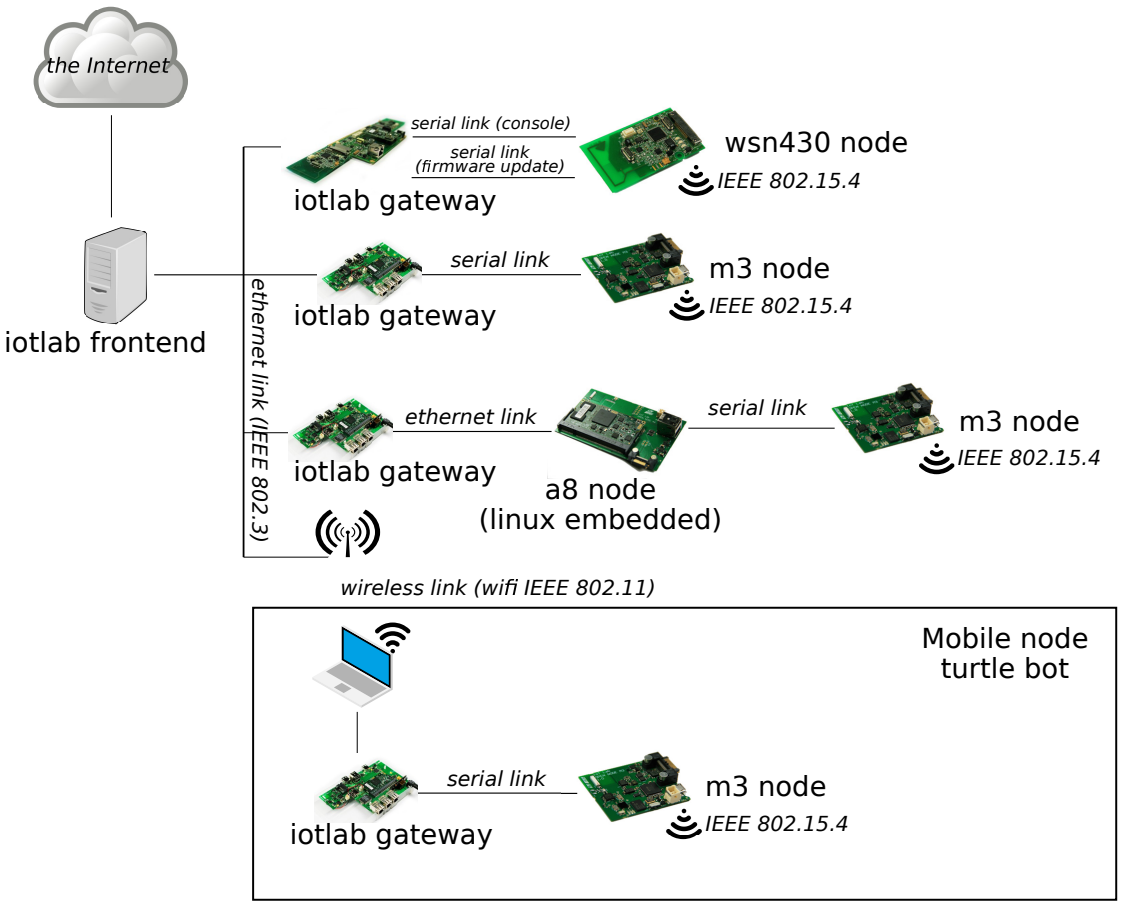


FIGURE 6.14 – Topologie d'un site donné.

Grid'5000. Les nœuds de la plateforme IoT-Lab ne sont pas voués à être utilisés en tant que nœud de stockage IPFS mais seulement en tant que client. Notre objectif est de développer une expérience qui interconnecte les deux plateforme : l'Internet des Capteurs avec l'Internet des Serveurs.

Afin de fournir une localité, nous proposons d'effectuer les expérimentations sur des sites physiques qui hébergent à la fois des machines Grid'5000 et des machines FIT, comme cela est le cas pour les sites de Grenoble, Lyon et Lille. Ces infrastructures, s'appuyant sur le réseau RENATER, nous pouvons nous attendre à un routage réseau direct entre les deux plateformes. Dans la suite, nous choisissons d'effectuer notre test à Grenoble. Nous avons développé un client IPFS pour le système d'exploitation RIOT, utilisable sur les capteurs de la plateforme IoT-Lab. Cela consiste en 900 lignes de code écrites en C et disponibles à l'adresse : https://github.com/bconfais/RIOT/tree/master/examples/ipfs_client. Une *pull-request* a également été soumise au projet officiel afin d'y faire intégrer ce programme[4].

[4] https://github.com/RIOT-OS/RIOT/pull/8889

6.3.3 Interconnexion des plateformes

Bien que les deux plateformes soient connectées au même opérateur réseau, celles-ci ne s'interconnectent que difficilement car chacune possède un réseau IPv4 non globalement routable, connecté au reste de l'Internet par une translation d'adresse (*NAT*) ou au travers d'une machine d'accès (*frontend*). Il n'est donc pas possible d'atteindre un nœud de la plateforme FIT depuis une machine Grid'5000 et inversement. De plus, Grid'5000 ne propose pas d'IPv6, rendant de fait, impossible une interconnexion avec ce protocole.

À partir des machines hébergées sur la plateforme Grid'5000, seule la machine d'accès de la plateforme FIT/IoT-Lab peut être contactée directement. Nous proposons donc d'établir un tunnel SSH entre ce *frontend* et une machine située sur la plateforme Grid'5000. Malheureusement, le *frontend* ne permet pas d'être joint depuis les autres machines de la plateforme. Nous établissons donc un second tunnel entre ce nœud et un nœud de type A8. Tout ceci est illustré dans la Figure 6.15.

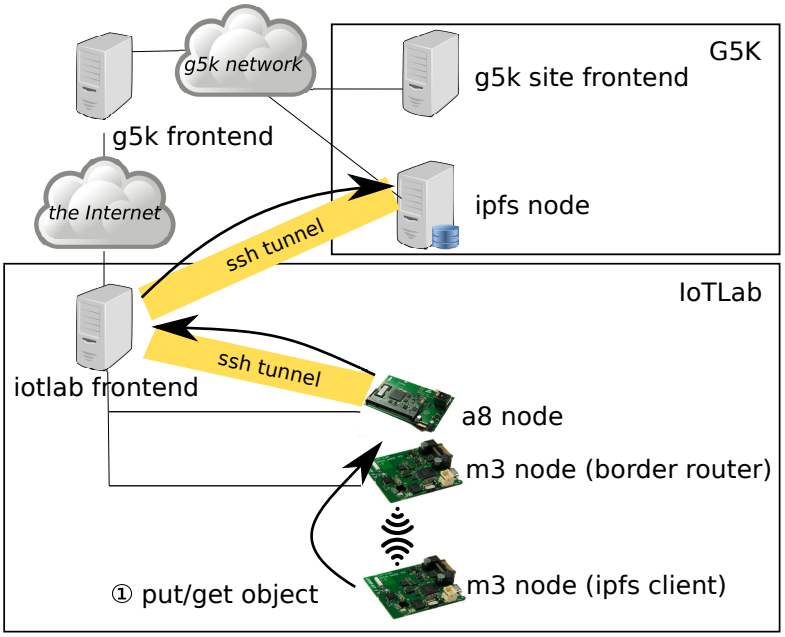


FIGURE 6.15 – Double tunnel établi entre les plateformes FIT/Iot-lab et Grid'5000.

Nous notons que Grid'5000 propose également un accès par un réseau privé virtuel (*Virtual Private Network, VPN*) mais la position de la machine de sortie ne permet pas de garantir un routage au plus court chemin, de réduire la latence réseau totale entre les deux plateformes et de confiner le trafic. En effet, cette machine est le point de passage obligé de tous les échanges entre les deux plateformes. Il est donc important qu'elle se trouve au même endroit physique que le site Grid'5000 et le site FIT/IoT-Lab pour ne pas dégrader la latence.

6.3.4 Limitations

Bien que l'interconnexion fonctionne, nous rencontrons de nombreuses limitations à son utilisation. Premièrement, développer un client IPFS pour les nœuds de la plateforme FIT n'est pas une chose simple. Bien que le système d'exploitation RIOT fournisse plusieurs piles réseau, aucune ne gère la fragmentation TCP. Cela limite nos requêtes à des objets de 80 octets pour l'instant.

Deuxièmement, le routage du tunnel n'est pas optimal, dans le sens où celui-ci n'a pas la latence la plus faible. Bien que les deux plateformes soient connectées à des réseaux gérés par le même opérateur et soient situés dans la même ville, notre tunnel est routé par Sophia, augmentant ainsi la latence totale.

Enfin, les latences sont fortement dégradées, non seulement à cause du tunnel mais aussi au sein de la plateforme FIT. Lorsque nous utilisons un nœud de type M3 connecté par lien radio, 30 ms sont nécessaires pour atteindre le nœud A8. Cette latence est notamment dégradée par le fait que le nœud M3 de bordure, faisant la liaison entre le réseau filaire et le réseau sans-fil, utilise une encapsulation des paquets IP à l'intérieur du lien série, utilisé pour contrôler le nœud. Cela est illustré par la Figure 6.16. Nous précisons que le débit entre le nœud A8 et le nœud de bordure M3 n'a pas été mesuré.

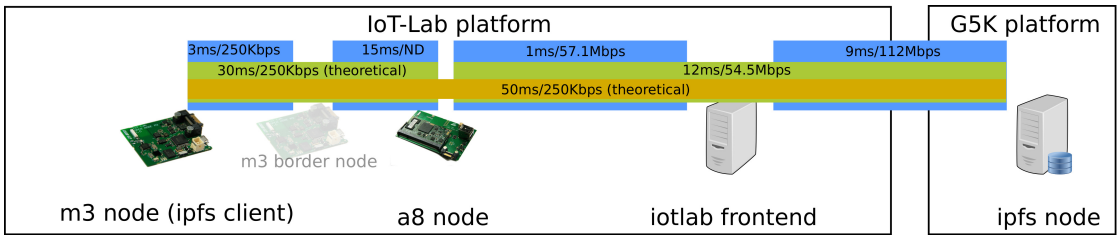


FIGURE 6.16 – Latences et débits mesurées lors de l'utilisation du tunnel entre la plateforme FIT/Iot-Lab et Grid'5000. Le débit entre le nœud A8 et le nœud M3 de bordure n'a pas été mesuré.

Malgré l'utilisation de tunnels pour interconnecter les plateformes et les limitations dûes à RIOT-OS, nous avons réussi, depuis le nœud M3, à écrire un objet sur la machine IPFS hébergée sur la plateforme Grid'5000. Bien que cela soit anecdotique, nous constatons que le temps d'écriture d'un seul objet de 80 octets est de l'ordre de 0, 72 seconde (soit un débit de 0.111 Ko/s). À titre de comparaison, nous avions mesuré un débit de 148 Ko/s lorsque nous écrivions les objets dans une infrastructure de type Cloud (1.72 seconde pour écrire 256 Ko d'après le Tableau 6.1(a)). Ce temps de 0,72 seconde n'est donc pas acceptable dans un environnement de Fog Computing où nous cherchons à réduire le plus possible les temps d'accès.

6.3.5 Conclusion

Utiliser la plateforme FIT/IoT-Lab en complément de la plateforme Grid'5000 pour réaliser une expérimentation dans un environnement de Fog plus réaliste n'est pas quelque

chose de facile, notamment parce que les deux plateformes sont difficiles à interconnecter mais aussi parce que les systèmes d'exploitations fournis par la plateforme IoT-Lab ne fournissent pas toutes les fonctionnalités auxquelles nous pouvons nous attendre, comme par exemple une pile TCP supportant la fragmentation.

Des investigations supplémentaires sont nécessaires pour corriger l'ensemble de ces difficultés. L'interconnexion des deux plateformes est l'objectif du projet SILECS[5] pour lequel, nous pouvons émettre les recommandations suivantes :

- Permettre des interconnexions IP directes entre les machines, que ce soit en IPv4 ou en IPv6 (pas de translation d'adresses) ;
- Avoir un routage au plus court chemin entre les deux plateformes ;
- Éviter d'encapsuler les connexions IP dans les liens série ;
- Mettre à disposition des machines de type raspberry-pi sur le même site que les capteurs serait également un plus puisque dans certains cas d'utilisation, ces machines peuvent fournir suffisamment de ressources. Cela peut également permettre d'exécuter des scénarios impliquant plusieurs niveaux de Fog.

6.4 Conclusion

Dans ce chapitre, nous avons évalué de façon expérimentale les différentes approches que nous avons proposées au chapitre précédent. Nous avons montré que le couplage d'IPFS avec un *Scale-Out NAS* permet de limiter les trafics inter-sites et d'émettre des données en dehors du site uniquement lorsque l'objet demandé n'est pas stocké localement, améliorant le confinement du trafic réseau mais aussi les temps d'accès. Contrairement à ce que nous pourrions penser, introduire un *Scale-Out NAS* n'introduit pas de surcoût dans le système lors des opérations d'écriture.

Lorsque les objets ne sont pas stockés localement et ont besoin d'être rapatriés, nous avons montré que notre stratégie de localisation reposant sur un arbre construit en respectant la topologie physique du réseau, permettait de localiser les objets plus rapidement en interrogeant en premier les sites proches en matière de latence. Cette stratégie permet non seulement de localiser les objets plus rapidement mais permet également de confiner le plus possible, le trafic émis entre les sites, au sein d'un voisinage restreint. Cela permet également d'accéder au réplica de données le plus proche. Enfin, dans une troisième partie, nous avons brièvement étudié la possibilité de réaliser des évaluations dans un environnement plus réaliste malgré de nombreuses difficultés (*i.e.*, Grid'5000 & FIT/IoT-Lab).

Ces évaluations sont l'une des premières à émuler un environnement de Fog Computing et confirment que nos propositions permettent à un système de stockage de fonctionner dans un tel environnement. Malgré ces résultats encourageants, nous pouvons avoir en perspective d'évaluer nos propositions dans d'autres scénarios, par exemple dans un

[5] http://silecs.net/

environnement avec plus de nœuds ou avec des nœuds insérés et retirés dynamiquement ou encore dans des environnements simulant des défaillances. Nous élaborons ces perspectives dans les derniers chapitres de ce manuscrit.

Conclusion générale

Dans ce travail, nous avons proposé un système de stockage par objets adapté à un environnement de Fog Computing, dans lequel les utilisateurs peuvent accéder rapidement à n'importe quelle donnée, quel que soit le site auquel ils sont connectés. Notre première partie a été consacrée à l'état de l'art tandis que nos contributions ont été détaillées dans une seconde partie.

Après avoir présenté un historique de l'informatique utilitaire, des grappes de calcul à l'informatique en nuage, nous avons introduit dans le Chapitre 1 les propriétés et les caractéristiques des infrastructures de Fog Computing, permettant de fournir de la puissance de calcul accessible avec une faible latence aux utilisateurs et aux objets connectés situés en bordure du réseau. Dans ce contexte, il est important de pouvoir gérer les ressources de calcul, de réseau et de stockage. Dans cette thèse, nous nous sommes donc concentrés sur le stockage de données dans les infrastructures distribuées. Nous avons ensuite, dans le Chapitre 2, détaillé le fonctionnement des principales solutions de stockage utilisées dans les grappes, les grilles de calcul mais aussi dans les infrastructures de Cloud Computing. Nous avons notamment souligné que plusieurs choix conceptuels, tels que la gestion centralisée des métadonnées n'étaient pas adaptés au contexte de cette thèse portant sur les infrastructures de Fog Computing.

Enfin, dans le Chapitre 3 qui termine cette partie sur l'état de l'art, nous avons discuté des caractéristiques de ce que nous attendions d'un système de stockage pour le Fog. Nous voulons *(i)* que les données soient stockées localement, sur le site le plus proche de l'utilisateur. Nous voulons également *(ii)* que le trafic réseau soit confiné, mais également *(iii)* que les données restent accessibles en cas de partitionnement de ce dernier. Enfin, la solution de stockage doit *(iv)* supporter la mobilité des utilisateurs entre les sites de Fog et *(v)* être utilisable avec un grand nombre de sites, d'utilisateurs et de données stockées. Nous avons également vérifié si, de par les concepts utilisés, si les solutions existantes comme Rados, Cassandra ou encore les solutions de stockage utilisées dans les réseaux de contenus pouvaient satisfaire ces propriétés.

Dans le Chapitre 4, notre première contribution a consisté à comparer expérimentalement, dans un environnement de Fog Computing, les performances de ces solutions, à savoir Rados, Cassandra et Interplanetary FileSystem (IPFS) qui est une solution de stockage se rapprochant le plus des solutions mises en œuvre dans les réseaux de diffusion de contenus. Il ressort de cette étude que Rados ne passe pas à l'échelle du fait de l'utilisation de l'algorithme Paxos permettant de gérer la consistance. Cassandra,

grâce à sa table de hachage distribuée à un saut, génère un trafic inter-sites réduit, qui ne dépend pas de la charge des nœuds. Nous avons cependant également montré que Cassandra n'est pas tolérant aux latences élevées. Enfin, IPFS, bien que fournissant de faibles temps d'accès, utilise une table de hachage distribuée pour localiser les objets, ce qui génère beaucoup de trafic inter-sites. Malgré les inconvénients de la solution de stockage IPFS, nous avons proposé dans le Chapitre 5, des améliorations visant à réduire le plus possible les échanges inter-sites. Nous avons tout d'abord proposé de coupler IPFS avec un système de stockage de type *Scale-Out NAS* déployé de façon indépendante sur chaque site. Tous les nœuds d'un site mettent ainsi en commun les objets qu'ils stockent, évitant d'utiliser la table de hachage distribuée pour localiser un objet stocké par un autre nœud du même site. De cette façon, les échanges inter-sites sont limités aux seuls accès à des objets distants.

Le processus de localisation des objets reposant sur une table de hachage distribuée a également mérité notre attention puisque les enregistrements de localisation sont répartis de façon uniforme grâce à une fonction de hachage, ce qui ne permet pas de confiner le trafic réseau. Ce manque de localité fait que pour télécharger un objet, il peut être plus coûteux d'accéder au site stockant la localisation qu'au site stockant l'objet lui-même. C'est pourquoi, dans une seconde section du Chapitre 5, nous avons proposé de remplacer la table de hachage distribuée par une approche utilisant un arbre, construit en respectant la topologie physique du réseau. Dans notre approche, des réplicas de localisation sont créés à la volée, lors de chaque accès, sur le chemin entre le site courant et l'endroit où la localisation fut trouvée. De cette manière, plus un objet est accédé, plus un réplica de localisation peut être trouvé avec une faible latence et un nombre réduit de sauts. Nous proposons également un algorithme permettant de construire l'arbre qui tient compte de notre façon itérative d'émettre les requêtes. Cet algorithme repose sur l'algorithme de Dijkstra dans lequel nous avons modifié la fonction de coût. Nous permettons également de générer des arbres qui ne sont pas optimaux en matière de latence, de façon à générer des arbres plus profonds dans lesquels les nœuds peuvent mieux bénéficier de la relocalisation.

Enfin, dans le dernier chapitre (Chapitre 6), nous avons évalué expérimentalement nos propositions, en émulant une infrastructure de Fog Computing sur la plateforme Grid'5000. Nous avons montré que coupler IPFS avec un système de stockage de type *Scale-Out NAS*, et en l'occurrence RozoFS, ne dégradait pas les performances en écriture. En lecture, en revanche, ne pas accéder à la DHT, améliore en moyenne de 34 % les temps d'accès lorsque l'objet accédé est un objet stocké sur le site local pour atteindre environ 2 secondes pour lire 100 objets de 10 Mo.

Nous avons par la suite évalué notre approche en arbre, à la fois sur des micro comparaisons puis sur une évaluation à plus large échelle. Nous avons montré que construire un réseau de recouvrement qui prend en compte la topologie physique et relocaliser les réplicas de localisation à la volée permet de réduire de 20 % les temps de localisation. Nous avons montré qu'avoir un arbre profond permettait d'augmenter les chances de localiser un objet en un faible nombre de sauts mais que dans le pire des cas, remonter jusqu'à la racine pour localiser un objet est une opération coûteuse.

Une dernière expérimentation a consisté à expérimenter la plateforme FIT/IoT-Lab

afin d'utiliser des capteurs comme clients de notre site de Fog. Cette expérimentation s'est révélée compliquée à mener, à la fois parce qu'utiliser les nœuds fournis sur la plateforme IoT-Lab demande d'importants efforts de développement mais surtout parce que les deux plateformes s'interconnectent difficilement du fait de l'absence de connexion IP de bout en bout.

Perspectives

Bien que notre solution de stockage soit fonctionnelle, de nombreux éléments peuvent être améliorés. Ces améliorations concernent à la fois l'aspect théorique de notre approche mais aussi son implémentation.

6.4.1 Amélioration de l'implémentation

Les performances de notre implémentation pourraient être améliorées par l'utilisation d'un protocole récursif. Au lieu que le nœud souhaitant accéder à l'objet envoie les requêtes aux différents nœuds, nous pouvons imaginer que si un nœud reçoit une requête pour un objet dont il ne connaît pas la localisation, il transmette celle-ci au nœud parent. Cela permettra de réduire le nombre de sauts et donc les temps d'accès. Des travaux sur cette problématique sont en cours, au sein du laboratoire LS2N (Alexandre van Kempen, post doc, équipe STACK).

6.4.2 Améliorations théoriques

Sur le plan théorique, une chose importante que nous n'avons pas traitée est le déploiement de l'arbre. Actuellement, l'arbre est calculé en amont. Cela peut avoir son importance si nous souhaitons gérer la dynamique du réseau, permettre aux sites de dynamiquement rejoindre le réseau, sans recourir à l'intervention d'un administrateur. Les approches que nous avons proposées ainsi que les évaluations ne tiennent pas compte de cela et l'arbre est manuellement construit par l'administrateur. Pourtant lorsque le réseau devient dynamique, recalculer l'arbre peut être une opération coûteuse en temps de calcul, ce qui nécessite de développer une approche adéquate. Bien que la DHT sache gérer nativement cette situation, la dynamicité du réseau peut être la source de beaucoup de trafic puisque un certain nombre de clés seront déplacées.

De plus, nous avon supposé une immuabilité des réplicas, dans nos contributions. Avoir un mécanisme permettant de garantir la consistance des données et d'invalider les réplicas des anciennes versions des objets semble nécessaire [Bravo et al. 2017]. Un tel mécanisme peut être difficile à trouver sans violer le théorème de Brewer stipulant que nous ne pouvons pas avoir simultanément la consistance des données, le support du mode déconnecté et la disponibilité des données (*i.e.*, le système répond dans un temps fini) [Brewer 2012].

La gestion des ressources au sein des sites de Fog Computing est également une autre perspective. Dans notre approche, nous avons supposé que les sites de Fog Computing avaient les ressources suffisantes pour traiter les demandes des utilisateurs et que les utilisateurs n'avaient qu'à se connecter au site le plus proche (dans le sens de la latence réseau). Pourtant dans la réalité, ceci n'est pas forcément vrai. Créer une stratégie permettant de supprimer les réplicas des objets mis en cache sur le site semble nécessaire. De la même manière une stratégie permettant d'associer chaque utilisateur à un site, en fonction de la quantité de données qu'il souhaite stocker et des ressources qu'il souhaite utiliser, pourrait être implémentée.

Déplacer les données ou créer de nouveaux réplicas de manière préventive est également une perspective à envisager. De nombreuses approches ont déjà été proposées dans ce sens [Sivasubramanian et al. 2005 ; Agarwal et al. 2010 ; Paiva et al. 2014]. Par exemple Agarwal *et al.* proposent de tenir compte des accès effectués afin d'optimiser le placement [Agarwal et al. 2010]. Cela permet de placer les réplicas « proches » des utilisateurs qui en ont besoin. Toutefois, cela est mis en œuvre par la résolution d'un problème d'optimisation linéaire, ce qui est coûteux en temps de calcul. C'est pourquoi, Paiva *et al.* proposent quant à eux, d'optimiser uniquement le placement des k objets les plus accédés [Paiva et al. 2014]. Les modèles prédictifs sont également une piste intéressante à considérer dans un contexte de l'Internet des Objets : l'idée est de prédire les sites où les données seront susceptibles d'être accédées afin d'y répliquer les données à l'avance [Gossa et al. 2008 ; Kosar et al. 2005 ; Baev et al. 2001].

Enfin, réfléchir à la façon dont sont réparties les données entre l'infrastructure de Cloud Computing, l'infrastructure de Fog et les clients est un sujet de recherche à part entière. Des algorithmes permettant de choisir si un calcul doit être effectué en bordure du réseau, dans l'infrastructure de Fog ou dans sur un serveur de Cloud Computing ont été proposés [Jararweh et al. 2016 ; Yousefpour et al. 2017]. Il pourrait également être envisagé d'utiliser une approche hybride, mélangeant les approche d'Extrême Edge Computing et de Fog Computing. Les clients d'un même site pourraient s'échanger des données sans passer par l'infrastructure de Fog, à la manière des approches CDN-P2P. Cela permettrait de réduire la charge de chaque site [Xu et al. 2006 ; Huang et al. 2008] qui ne seraient sollicités que lorsque la donnée demandée n'est possédée par aucun client du site. Une telle stratégie permettant de localiser les données parmi les clients est à développer.

6.4.3 Perspectives de déploiement et d'utilisation

Il n'existe pas encore à l'heure actuelle de déploiement industriel à large échelle d'architecture de Fog Computing et nous ne savons pas encore si le concept sera réellement utilisé par les entreprises et pour quels usages. Néanmoins, de nombreux acteurs, à la fois industriels et académiques regroupés au sein de l'Open Fog Consortium[6] travaillent à son déploiement. La sécurité est l'un des principaux aspects aujourd'hui considérer, notamment pour garantir l'intégrité des données et des calculs. De la même façon que le

[6] https://www.openfogconsortium.org/

protocole DNS a inspiré notre approche pour localiser l'objet, son extension DNSSEC peut être une source d'inspiration.

Aujourd'hui, certains opérateurs réseau ont l'impression de ne servir que d'intermédiaire entre les utilisateurs et les entreprises possédant les infrastructures de Cloud Computing. Ils espèrent que le Fog Computing sera pour eux l'occasion de développer leur activité, notamment en fournissant aux utilisateurs des services que les entreprises comme Google ou Amazon ne pourront que difficilement fournir à cause des contraintes de latence. Pourtant, des services comme *Amazon Lambda@Edge*[7] ou Akamai Cloudlet [Pang et al. 2015] voient le jour, accentuant la concurrence entre les opérateurs réseau et les fournisseurs de services.

Des perspectives d'utilisation existent dans de nombreux domaines. Par exemple, dans les « villes intelligentes », le Fog Computing peut être utilisé afin de réguler le trafic routier, optimiser les flux logistiques et la consommation énergétique [Katsak et al. 2015 ; Faruque et al. 2016].

Toutefois, prendre en compte une telle architecture lors du développement d'un logiciel n'est pas chose aisée et plusieurs articles proposent des modèles de programmation [Hong et al. 2013 ; Giang et al. 2015] pour développer des programmes qui fonctionnent sur les infrastructures de Fog Computing et qui tiennent compte de leur spécificités. Le succès des infrastructures de Fog Computing dépendra donc non seulement de la capacité par les industriels de déployer les sites en nombre suffisants, proche des utilisateurs mais surtout de la capacité et la volonté des développeurs d'utiliser de telles infrastructures. La facilité de développement sera certainement l'un des critères déterminants qui fera du Fog Computing, une infrastructure populaire et largement utilisée.

[7] https://docs.aws.amazon.com/lambda/latest/dg/lambda-edge.html

Références bibliographiques

Aazam, Mohammad et Eui-Nam Huh (2014). « Fog Computing and Smart Gateway Based Communication for Cloud of Things ». Dans : *Proceedings of the 2014 International Conference on Future Internet of Things and Cloud*. FICLOUD '14. Washington, DC, USA : IEEE Computer Society, p. 464-470. ISBN : 978-1-4799-4357-9. DOI : 10.1109/FiCloud.2014.83.

Adjih, C., E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele et T. Watteyne (déc. 2015). « FIT IoT-Lab : A large scale open experimental IoT testbed ». Dans : *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, p. 459-464. DOI : 10.1109/WF-IoT.2015.7389098.

Agarwal, Sharad, John Dunagan, Navendu Jain, Stefan Saroiu, Alec Wolman et Harbinder Bhogan (2010). « Volley : Automated Data Placement for Geo-distributed Cloud Services ». Dans : *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*. NSDI'10. San Jose, California : USENIX Association, p. 2-2.

Agrawal, Divyakant, Sudipto Das et Amr El Abbadi (2011). « Big Data and Cloud Computing : Current State and Future Opportunities ». Dans : *Proceedings of the 14th International Conference on Extending Database Technology*. EDBT/ICDT '11. Uppsala, Sweden : ACM, p. 530-533. ISBN : 978-1-4503-0528-0. DOI : 10.1145/1951365.1951432.

Ahmed, A. et E. Ahmed (jan. 2016). « A survey on mobile edge computing ». Dans : *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, p. 1-8. DOI : 10.1109/ISCO.2016.7727082.

Alpert, C. J., T. C. Hu, J. H. Huang et A. B. Kahng (mai 1993). « A direct combination of the Prim and Dijkstra constructions for improved performance-driven global routing ». Dans : *1993 IEEE International Symposium on Circuits and Systems*, 1869-1872 vol.3. DOI : 10.1109/ISCAS.1993.394112.

Anderson, J. Chris, Jan Lehnardt et Noah Slater (2010). *CouchDB : The Definitive Guide Time to Relax*. 1st. O'Reilly Media, Inc. ISBN : 0596155891, 9780596155896.

Anwar, Ali, Yue Cheng, Aayush Gupta et Ali R. Butt (2016). « MOS : Workload-aware Elasticity for Cloud Object Stores ». Dans : *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*. HPDC '16. Kyoto, Japan : ACM, p. 177-188. isbn : 978-1-4503-4314-5. doi : 10.1145/2907294.2907304.

Ashraf, K., R. Anane et B. Bordbar (mar. 2012). « File Management in a Mobile DHT-based P2P Environment ». Dans : *2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, p. 415-422. doi : 10.1109/AINA.2012.95.

Atikoglu, Berk, Yuehai Xu, Eitan Frachtenberg, Song Jiang et Mike Paleczny (juin 2012). « Workload Analysis of a Large-scale Key-value Store ». Dans : *SIGMETRICS Perform. Eval. Rev.* 40.1, p. 53-64. issn : 0163-5999. doi : 10.1145/2318857.2254766.

Baccelli, E., O. Hahm, M. Gunes, M. Wahlisch et T. C. Schmidt (avr. 2013). « RIOT OS : Towards an OS for the Internet of Things ». Dans : *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, p. 79-80. doi : 10.1109/INFCOMW.2013.6970748.

Baev, Ivan, Rajmohan Rajaraman et Chaitanya Swamy (août 2008). « Approximation Algorithms for Data Placement Problems ». Dans : *SIAM J. Comput.* 38.4, p. 1411-1429. issn : 0097-5397. doi : 10.1137/080715421.

Baev, Ivan D. et Rajmohan Rajaraman (2001). « Approximation Algorithms for Data Placement in Arbitrary Networks ». Dans : *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '01. Washington, D.C., USA : Society for Industrial et Applied Mathematics, p. 661-670. isbn : 0-89871-490-7.

Balouek, Daniel, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lebre, David Margery, Nicolas Niclausse, Lucas Nussbaum, Olivier Richard, Christian Pérez, Flavien Quesnel, Cyril Rohr et Luc Sarzyniec (2013). « Adding Virtualization Capabilities to the Grid'5000 Testbed ». Dans : *Cloud Computing and Services Science*. Sous la dir. d'IvanI. Ivanov, Marten Sinderen, Frank Leymann et Tony Shan. T. 367. Communications in Computer and Information Science. Springer International Publishing, p. 3-20. isbn : 978-3-319-04518-4. doi : 10.1007/978-3-319-04519-1_1.

Barry, R. (2010). *Using the FreeRTOS Real Time Kernel : A Practical Guide*. Real Time Engineers Limited. isbn : 9781446169148.

Beame, Carl, Robert Thurlow, Brent Callaghan, David Robinson, David Noveck, Mike Eisler et Spencer Shepler (avr. 2003). *Network File System (NFS) version 4 Protocol*. RFC 3530. doi : 10.17487/RFC3530.

Beck, Michael Till, Martin Werner, Sebastian Feld et Thomas Schimper (2014). « Mobile Edge Computing : A Taxonomy ». Dans : *AFIN : The Sixth International Conference on Advances in Future Internet*. Citeseer.

Benet, Juan (2014). *IPFS - Content Addressed, Versioned, P2P File System*. Rapp. tech. Protocol Labs, Inc.

Biswas, A. R. et R. Giaffreda (mar. 2014). « IoT and cloud convergence : Opportunities and challenges ». Dans : *2014 IEEE World Forum on Internet of Things (WF-IoT)*, p. 375-376. doi : 10.1109/WF-IoT.2014.6803194.

Bittencourt, L. F., M. M. Lopes, I. Petri et O. F. Rana (nov. 2015). « Towards Virtual Machine Migration in Fog Computing ». Dans : *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, p. 1-8. doi : 10.1109/3PGCIC.2015.85.

Bittencourt, L. F., J. Diaz-Montes, R. Buyya, O. F. Rana et M. Parashar (mar. 2017). « Mobility-Aware Application Scheduling in Fog Computing ». Dans : *IEEE Cloud Computing* 4.2, p. 26-35. issn : 2325-6095. doi : 10.1109/MCC.2017.27.

Blagodurov, Sergey, Alexandra Fedorova, Evgeny Vinnik, Tyler Dwyer et Fabien Hermenier (2015). « Multi-objective Job Placement in Clusters ». Dans : *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '15. Austin, Texas : ACM, 66 :1-66 :12. isbn : 978-1-4503-3723-6. doi : 10.1145/2807591.2807636.

Bonomi, Flavio, Rodolfo Milito, Jiang Zhu et Sateesh Addepalli (2012). « Fog Computing and Its Role in the Internet of Things ». Dans : *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. MCC '12. Helsinki, Finland : ACM, p. 13-16. isbn : 978-1-4503-1519-7. doi : 10.1145/2342509.2342513.

Bonomi, Flavio, Rodolfo Milito, Preethi Natarajan et Jiang Zhu (2014). « Fog Computing : A Platform for Internet of Things and Analytics ». Dans : *Big Data and Internet of Things : A Roadmap for Smart Environments*. Sous la dir. de Nik Bessis et Ciprian Dobre. Cham : Springer International Publishing, p. 169-186. isbn : 978-3-319-05029-4. doi : 10.1007/978-3-319-05029-4_7.

Borg, J (2015). *SyncThing (2015)*. Rapp. tech. Syncthing Inc.

Bravo, Manuel, Luís Rodrigues et Peter Van Roy (2017). « Saturn : A Distributed Metadata Service for Causal Consistency ». Dans : *Proceedings of the Twelfth European Conference on Computer Systems*. EuroSys '17. Belgrade, Serbia : ACM, p. 111-126. isbn : 978-1-4503-4938-3. doi : 10.1145/3064176.3064210.

Brewer, E. (fév. 2012). « CAP twelve years later : How the "rules" have changed ». Dans : *Computer* 45.2, p. 23-29. issn : 0018-9162. doi : 10.1109/MC.2012.37.

Brito, M. S. D., S. Hoque, R. Steinke et A. Willner (sept. 2016). « Towards Programmable Fog Nodes in Smart Factories ». Dans : *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, p. 236-241. doi : 10.1109/FAS-W.2016.57.

Bronson, Nathan, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, Mark Marchukov, Dmitri Petrov, Lovro Puzar, Yee Jiun Song et Venkat Venkataramani (2013). « TAO : Facebook's Distributed Data Store for the Social Graph ». Dans : *Proceedings of the 2013 USENIX Conference on Annual Technical Conference*. USENIX ATC'13. San Jose, CA : USENIX Association, p. 49-60.

Byers, Charles C. et Patrick Wetterwald (nov. 2015). « Fog Computing Distributing Data and Intelligence for Resiliency and Scale Necessary for IoT : The Internet of Things (Ubiquity Symposium) ». Dans : *Ubiquity* 2015.November, 4 :1-4 :12. issn : 1530-2180. doi : 10.1145/2822875.

Cao, Yu, Songqing Chen, Peng Hou et D. Brown (août 2015). « FAST : A fog computing assisted distributed analytics system to monitor fall for stroke mitigation ». Dans : *2015 IEEE International Conference on Networking, Architecture and Storage (NAS)*, p. 2-11. doi : 10.1109/NAS.2015.7255196.

Carlson, Josiah L. (2013). *Redis in Action*. Greenwich, CT, USA : Manning Publications Co. isbn : 1617290858, 9781617290855.

Carns, Philip H., Walter B. Ligon III, Robert B. Ross et Rajeev Thakur (2000). « PVFS : A Parallel File System for Linux Clusters ». Dans : *Proceedings of the 4th Annual Linux Showcase & Conference - Volume 4*. ALS'00. Atlanta, Georgia : USENIX Association, p. 28-28.

Casado, Martin, Teemu Koponen, Scott Shenker et Amin Tootoonchian (2012). « Fabric : A Retrospective on Evolving SDN ». Dans : *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. HotSDN '12. Helsinki, Finland : ACM, p. 85-90. isbn : 978-1-4503-1477-0. doi : 10.1145/2342441.2342459.

Castro, Miguel, Peter Druschel, Y. Charlie Hu et Antony Rowstron (2003). « Topology-Aware Routing in Structured Peer-to-Peer Overlay Networks ». Dans : *Future Directions in Distributed Computing : Research and Position Papers*. Sous la dir. d'André Schiper, Alex A. Shvartsman, Hakim Weatherspoon et Ben Y. Zhao. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 103-107. isbn : 978-3-540-37795-5. doi : 10.1007/3-540-37795-6_19.

Chandy, John A. (2008). « A Generalized Replica Placement Strategy to Optimize Latency in a Wide Area Distributed Storage System ». Dans : *Proceedings of the 2008 International Workshop on Data-aware Distributed Computing*. DADC '08. Boston, MA, USA : ACM, p. 49-54. isbn : 978-1-60558-154-5. doi : 10.1145/1383519.1383525.

Chen, Jianjun, Chris Douglas, Michi Mutsuzaki, Patrick Quaid, Raghu Ramakrishnan, Sriram Rao et Russell Sears (2012). « Walnut : A Unified Cloud Object Store ». Dans : *Proceedings of the 2012 ACM SIGMOD International Conference on*

Management of Data. SIGMOD '12. Scottsdale, Arizona, USA : ACM, p. 743-754. ISBN : 978-1-4503-1247-9. DOI : 10.1145/2213836.2213947.

CHEN, Xin, Haining WANG et Shansi REN (2006). « DNScup : Strong Cache Consistency Protocol for DNS ». Dans : *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*. ICDCS '06. Washington, DC, USA : IEEE Computer Society, p. 40-. ISBN : 0-7695-2540-7. DOI : 10.1109/ICDCS.2006.31.

CHIANG, M. et T. ZHANG (déc. 2016). « Fog and IoT : An Overview of Research Opportunities ». Dans : *IEEE Internet of Things Journal* 3.6, p. 854-864. ISSN : 2327-4662. DOI : 10.1109/JIOT.2016.2584538.

CLAUSEN, Thomas H. et Philippe JACQUET (oct. 2003). *Optimized Link State Routing Protocol (OLSR)*. RFC 3626. DOI : 10.17487/RFC3626.

CONFAIS, Bastien, Alexandre VAN KEMPEN, Sylvain DAVID, Benoît PARREIN et Marie-Pierre NACHOUKI (juin 2015). « Distributed Filesystems comparison on the GRID'5000 cluster ». Dans : *Third Sino-French Workshop on Information and Communications Technology (SFWICT)*. Nantes, France.

CONFAIS, Bastien, Adrien LEBRE et Benoît PARREIN (déc. 2016a). « Performance Analysis of Object Store Systems in a Fog/Edge Computing Infrastructures ». Dans : *IEEE CloudCom*. Luxembourg, Luxembourg.

CONFAIS, Bastien, Adrien LEBRE et Benoît PARREIN (mai 2017b). « An Object Store Service for a Fog/Edge Computing Infrastructure based on IPFS and Scale-out NAS ». Dans : *1st IEEE International Conference on Fog and Edge Computing - ICFEC'2017*. Madrid, Spain.

CONFAIS, Bastien, Adrien LEBRE et Benoît PARREIN (août 2017d). « Performance Analysis of Object Store Systems in a Fog and Edge Computing Infrastructure ». Dans : *Transactions on Large-Scale Data- and Knowledge-Centered Systems XXXIII*. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 40-79. ISBN : 978-3-662-55696-2. DOI : 10.1007/978-3-662-55696-2_2.

CONFAIS, Bastien, Adrien LEBRE et Benoît PARREIN (2018a). « A Tree-Based Approach to locate Object Replicas in a Fog Storage Infrastructure (en soumission) ». Dans : *IEEE Global Communications Conference (GlobeCom 2018)*.

CONFAIS, Bastien, Adrien LEBRE et Benoît PARREIN (jan. 2018c). « Adaptation of the Dijkstra's algorithm for metadata management in Fog Computing ». Dans : *Journées non thématiques, RESCOM 2018*. Toulouse, France.

CONFAIS, Bastien, Adrien LEBRE et Benoît PARREIN (avr. 2018e). « Improving locality of an object store working in a Fog environment ». Dans : *1st Grid'5000-FIT school*. Nice, France.

COOPER, Brian F., Adam SILBERSTEIN, Erwin TAM, Raghu RAMAKRISHNAN et Russell SEARS (2010). « Benchmarking Cloud Serving Systems with YCSB ». Dans :

Proceedings of the 1st ACM Symposium on Cloud Computing. SoCC '10. Indianapolis, Indiana, USA : ACM, p. 143-154. ISBN : 978-1-4503-0036-0. DOI : 10.1145/1807128.1807152.

COSTA-REQUENA, J., J. L. SANTOS, V. F. GUASCH, K. AHOKAS, G. PREMSANKAR, S. LUUKKAINEN, O. L. PÉREZ, M. U. ITZAZELAIA, I. AHMAD, M. LIYANAGE, M. YLIANTTILA et E. M. de OCA (juin 2015). « SDN and NFV integration in generalized mobile network architecture ». Dans : *2015 European Conference on Networks and Communications (EuCNC)*, p. 154-158. DOI : 10.1109/EuCNC.2015.7194059.

COUTO, R. D. Souza, S. SECCI, M. E. Mitre CAMPISTA et L. H. Maciel Kosmalski COSTA (oct. 2014). « Network design requirements for disaster resilience in IaaS clouds ». Dans : *IEEE Communications Magazine* 52.10, p. 52-58. ISSN : 0163-6804. DOI : 10.1109/MCOM.2014.6917402.

COX, Russ, Athicha MUTHITACHAROEN et Robert T. MORRIS (2002). « Serving DNS Using a Peer-to-Peer Lookup Service ». Dans : *Peer-to-Peer Systems*. Sous la dir. de Peter DRUSCHEL, Frans KAASHOEK et Antony ROWSTRON. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 155-165. ISBN : 978-3-540-45748-0.

CROMAN, Kyle, Christian DECKER, Ittay EYAL, Adem Efe GENCER, Ari JUELS, Ahmed KOSBA, Andrew MILLER, Prateek SAXENA, Elaine SHI, Emin GÜN SIRER, Dawn SONG et Roger WATTENHOFER (2016). « On Scaling Decentralized Blockchains ». Dans : *Financial Cryptography and Data Security*. Sous la dir. de Jeremy CLARK, Sarah MEIKLEJOHN, Peter Y.A. RYAN, Dan WALLACH, Michael BRENNER et Kurt ROHLOFF. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 106-125. ISBN : 978-3-662-53357-4.

DABEK, Frank, M. Frans KAASHOEK, David KARGER, Robert MORRIS et Ion STOICA (oct. 2001). « Wide-area Cooperative Storage with CFS ». Dans : *SIGOPS Oper. Syst. Rev.* 35.5, p. 202-215. ISSN : 0163-5980. DOI : 10.1145/502059.502054.

DABEK, Frank, Jinyang LI, Emil SIT, James ROBERTSON, M. Frans KAASHOEK et Robert MORRIS (2004a). « Designing a DHT for Low Latency and High Throughput ». Dans : *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*. NSDI'04. San Francisco, California : USENIX Association, p. 7-7.

DABEK, Frank, Russ COX, Frans KAASHOEK et Robert MORRIS (août 2004b). « Vivaldi : A Decentralized Network Coordinate System ». Dans : *SIGCOMM Comput. Commun. Rev.* 34.4, p. 15-26. ISSN : 0146-4833. DOI : 10.1145/1030194.1015471.

DANNEN, Chris (2017). *Introducing Ethereum and Solidity : Foundations of Cryptocurrency and Blockchain Programming for Beginners*. 1st. Berkely, CA, USA : Apress. ISBN : 1484225341, 9781484225349.

Dastjerdi, Amir Vahid et Rajkumar Buyya (août 2016a). « Fog Computing : Helping the Internet of Things Realize Its Potential ». Dans : *Computer* 49.8, p. 112-116. issn : 0018-9162. doi : 10.1109/MC.2016.245.

Dastjerdi, A.V., H. Gupta, R.N. Calheiros, S.K. Ghosh et R. Buyya (2016b). « Chapter 4 - Fog Computing : Principles, Architectures, and Applications ». Dans : *Internet of Things*. Sous la dir. de Rajkumar Buyya et Amir Vahid Dastjerdi. Morgan Kaufmann, p. 61 -75. isbn : 978-0-12-805395-9. doi : 10.1016/B978-0-12-805395-9.00004-6.

Davies, Alex et Alessandro Orsaria (nov. 2013). « Scale out with GlusterFS ». Dans : *Linux J.* 2013.235. issn : 1075-3583.

DeCandia, Giuseppe, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall et Werner Vogels (oct. 2007). « Dynamo : Amazon's Highly Available Key-value Store ». Dans : *SIGOPS Oper. Syst. Rev.* 41.6, p. 205-220. issn : 0163-5980. doi : 10.1145/1323293.1294281.

Dey, Swarnava et Arijit Mukherjee (2016). « Robotic SLAM : A Review from Fog Computing and Mobile Edge Computing Perspective ». Dans : *Adjunct Proceedings of the 13th International Conference on Mobile and Ubiquitous Systems : Computing Networking and Services*. MOBIQUITOUS 2016. Hiroshima, Japan : ACM, p. 153-158. isbn : 978-1-4503-4759-4. doi : 10.1145/3004010.3004032.

Dijkstra, E. W. (déc. 1959). « A Note on Two Problems in Connexion with Graphs ». Dans : *Numer. Math.* 1.1, p. 269-271. issn : 0029-599X. doi : 10.1007/BF01386390.

Dimakis, A. G., P. B. Godfrey, Y. Wu, M. J. Wainwright et K. Ramchandran (sept. 2010). « Network Coding for Distributed Storage Systems ». Dans : *IEEE Transactions on Information Theory* 56.9, p. 4539-4551. issn : 0018-9448. doi : 10.1109/TIT.2010.2054295.

Doi, Y. (jan. 2005). « DNS meets DHT : treating massive ID resolution using DNS over DHT ». Dans : *The 2005 Symposium on Applications and the Internet*, p. 9-15. doi : 10.1109/SAINT.2005.22.

Donassolo, Bruno, Ilhem Fajjari, Arnaud Legrand et Mertikopoulos Panayotis (avr. 2018). « FogIoT Orchestrator : an Orchestration System for IoT Applications in Fog Environment ». Dans : *1st Grid'5000-FIT school*. Nice, France.

Donovan, S, G Huizenga, AJ Hutton, CC Ross, MK Petersen et P Schwan (2003). « Lustre : Building a file system for 1000-node clusters ». Dans : *Proceedings of the Linux Symposium*.

Dubey, Harishchandra, Jing Yang, Nick Constant, Amir Mohammad Amiri, Qing Yang et Kunal Makodiya (2015). « Fog Data : Enhancing Telehealth Big Data Through Fog Computing ». Dans : *Proceedings of the ASE BigData ; SocialInformatics*

2015. ASE BD ;SI '15. Kaohsiung, Taiwan : ACM, 14 :1-14 :6. ISBN : 978-1-4503-3735-9. DOI : 10.1145/2818869.2818889.

DUNKELS, Adam, Oliver SCHMIDT, Niclas FINNE, Joakim ERIKSSON, Fredrik ÖSTERLIND, Nicolas TSIFTES et Mathilde DURVY (2011). *The contiki os : The operating system for the internet of things*. Rapp. tech. Thingsquare AB.

EL DICK, Manal, Esther PACITTI et Bettina KEMME (2009). « Flower-CDN : A Hybrid P2P Overlay for Efficient Query Processing in CDN ». Dans : *Proceedings of the 12th International Conference on Extending Database Technology : Advances in Database Technology*. EDBT '09. Saint Petersburg, Russia : ACM, p. 427-438. ISBN : 978-1-60558-422-5. DOI : 10.1145/1516360.1516410.

EUM, S., M. JIBIKI, M. MURATA, H. ASAEDA et N. NISHINAGA (juil. 2015). « A design of an ICN architecture within the framework of SDN ». Dans : *2015 Seventh International Conference on Ubiquitous and Future Networks*, p. 141-146. DOI : 10.1109/ICUFN.2015.7182521.

EVANS, Dave (2011). *The internet of things : How the next evolution of the internet is changing everything*. Rapp. tech. 2011. CISCO white paper, p. 1-11.

FAN, Xun, Ethan KATZ-BASSETT et John HEIDEMANN (2015). « Assessing Affinity Between Users and CDN Sites ». Dans : *Traffic Monitoring and Analysis*. Sous la dir. de Moritz STEINER, Pere BARLET-ROS et Olivier BONAVENTURE. Cham : Springer International Publishing, p. 95-110. ISBN : 978-3-319-17172-2.

FANTAR, Sonia Gaied et Habib YOUSSEF (2009). « Locality-aware Chord over Mobile Ad Hoc Networks ». Dans : *Proceedings of the Second International Conference on Global Information Infrastructure Symposium*. GIIS'09. Hammamet, Tunisia : IEEE Press, p. 136-141. ISBN : 978-1-4244-4623-0.

FARINA, Jason, Mark SCANLON et M-Tahar KECHADI (2014). « BitTorrent Sync : First Impressions and Digital Forensic Implications ». Dans : *Digital Investigation* 11. Proceedings of the First Annual DFRWS Europe, S77 -S86. ISSN : 1742-2876. DOI : 10.1016/j.diin.2014.03.010.

FARUQUE, M. A. Al et K. VATANPARVAR (avr. 2016). « Energy Management-as-a-Service Over Fog Computing Platform ». Dans : *IEEE Internet of Things Journal* 3.2, p. 161-169. ISSN : 2327-4662. DOI : 10.1109/JIOT.2015.2471260.

FAYAZBAKHSH, Seyed Kaveh, Yin LIN, Amin TOOTOONCHIAN, Ali GHODSI, Teemu KOPONEN, Bruce MAGGS, K.C. NG, Vyas SEKAR et Scott SHENKER (août 2013). « Less Pain, Most of the Gain : Incrementally Deployable ICN ». Dans : *SIGCOMM Comput. Commun. Rev.* 43.4, p. 147-158. ISSN : 0146-4833. DOI : 10.1145/2534169.2486023.

FERNANDO, Niroshinie, Seng W. LOKE et Wenny RAHAYU (2013). « Mobile cloud computing : A survey ». Dans : *Future Generation Computer Systems* 29.1. Including

Special section : AIRCC-NetCoM 2009 and Special section : Clouds and Service-Oriented Architectures, p. 84 -106. ISSN : 0167-739X. DOI : 10.1016/j.future.2012.05.023.

FIRDHOUS, Mohamed, Osman GHAZALI et Suhaidi HASSAN (2014). « Fog Computing : Will it be the Future of Cloud Computing ? » Dans : *Third International Conference on Informatics & Applications, Kuala Terengganu, Malaysia*, p. 8-15. ISBN : 978-1-941968-00-0.

FITFIELD, Tom (nov. 2013). « Introduction to OpenStack ». Dans : *Linux J.* 2013.235. ISSN : 1075-3583.

FREEDMAN, Michael J., Eric FREUDENTHAL et David MAZIÈRES (2004). « Democratizing Content Publication with Coral ». Dans : *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*. NSDI'04. San Francisco, California : USENIX Association, p. 18-18.

GANESAN, Prasanna, Krishna GUMMADI et H. GARCIA-MOLINA (2004). « Canon in G major : designing DHTs with hierarchical structure ». Dans : *24th International Conference on Distributed Computing Systems, 2004. Proceedings.* P. 263-272. DOI : 10.1109/ICDCS.2004.1281591.

GANGULY, S., A. SAXENA, S. BHATNAGAR, R. IZMAILOV et S. BANERJEE (mar. 2005). « Fast replication in content distribution overlays ». Dans : *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.* T. 4, 2246-2256 vol. 4. DOI : 10.1109/INFCOM.2005.1498512.

GAO, Peter Xiang, Andrew R. CURTIS, Bernard WONG et Srinivasan KESHAV (2012). « It's Not Easy Being Green ». Dans : *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM '12. Helsinki, Finland : ACM, p. 211-222. ISBN : 978-1-4503-1419-0. DOI : 10.1145/2342356.2342398.

GARCÉS-ERICE, L., E. W. BIERSACK, P. A. FELBER, K. W. ROSS et G. URVOY-KELLER (2003). « Hierarchical Peer-to-Peer Systems ». Dans : *Euro-Par 2003 Parallel Processing*. Sous la dir. d'Harald KOSCH, László BÖSZÖRMÉNYI et Hermann HELLWAGNER. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 1230-1239. ISBN : 978-3-540-45209-6.

GARCÍA-BARRIOCANAL, Elena, Salvador SÁNCHEZ-ALONSO et Miguel-Angel SICILIA (2017). « Deploying Metadata on Blockchain Technologies ». Dans : *Metadata and Semantic Research*. Sous la dir. d'Emmanouel GAROUFALLOU, Sirje VIRKUS, Rania SIATRI et Damiana KOUTSOMIHA. Cham : Springer International Publishing, p. 38-49. ISBN : 978-3-319-70863-8.

GARCIA LOPEZ, Pedro, Alberto MONTRESOR, Dick EPEMA, Anwitaman DATTA, Teruo HIGASHINO, Adriana IAMNITCHI, Marinho BARCELLOS, Pascal FELBER et Etienne RIVIERE (sept. 2015). « Edge-centric Computing : Vision and Challenges ». Dans :

SIGCOMM Comput. Commun. Rev. 45.5, p. 37-42. ISSN : 0146-4833. DOI : 10.1145/2831347.2831354.

GETTYS, Jim et Kathleen NICHOLS (jan. 2012). « Bufferbloat : Dark Buffers in the Internet ». Dans : *Commun. ACM* 55.1, p. 57-65. ISSN : 0001-0782. DOI : 10.1145/2063176.2063196.

GHAREEB, Majd, Soufiane ROUIBIA, Benoît PARREIN, Mohamad RAAD et Cedric THAREAU (juin 2013). « P2PWeb : a Client/Server and P2P Hybrid Architecture for Content Delivery over Internet ». Dans : *Third International Conference on Communications and Information Technology ICCIT 2013*. Beirut, Lebanon, pp.1-5.

GIANG, N. K., M. BLACKSTOCK, R. LEA et V. C. M. LEUNG (oct. 2015). « Developing IoT applications in the Fog : A Distributed Dataflow approach ». Dans : *2015 5th International Conference on the Internet of Things (IOT)*, p. 155-162. DOI : 10.1109/IOT.2015.7356560.

GIFFORD, David K., Roger M. NEEDHAM et Michael D. SCHROEDER (mar. 1988). « The Cedar File System ». Dans : *Commun. ACM* 31.3, p. 288-298. ISSN : 0001-0782. DOI : 10.1145/42392.42398.

GILL, Phillipa, Martin ARLITT, Zongpeng LI et Anirban MAHANTI (2007). « Youtube Traffic Characterization : A View from the Edge ». Dans : *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*. IMC '07. San Diego, California, USA : ACM, p. 15-28. ISBN : 978-1-59593-908-1. DOI : 10.1145/1298306.1298310.

GIURGIU, Ioana, Oriana RIVA, Dejan JURIC, Ivan KRIVULEV et Gustavo ALONSO (2009). « Calling the Cloud : Enabling Mobile Phones As Interfaces to Cloud Applications ». Dans : *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*. Middleware '09. Urbanna, Illinois : Springer-Verlag New York, Inc., 5 :1-5 :20.

GKANTSIDIS, C., M. MIHAIL et A. SABERI (mar. 2004). « Random walks in peer-to-peer networks ». Dans : *IEEE INFOCOM 2004*. T. 1, p. 130. DOI : 10.1109/INFCOM.2004.1354487.

GOSSA, J., A. G. JANECEK, K. A. HUMMEL, W. N. GANSTERER et J. M. PIERSON (avr. 2008). « Proactive Replica Placement Using Mobility Prediction ». Dans : *2008 Ninth International Conference on Mobile Data Management Workshops, MDMW*, p. 182-189. DOI : 10.1109/MDMW.2008.21.

GUBBI, Jayavardhana, Rajkumar BUYYA, Slaven MARUSIC et Marimuthu PALANISWAMI (2013). « Internet of Things (IoT) : A vision, architectural elements, and future directions ». Dans : *Future Generation Computer Systems* 29.7. Including Special sections : Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services & Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond, p. 1645 -1660. ISSN : 0167-739X. DOI : 10.1016/j.future.2013.01.010.

Hahm, Oliver, Emmanuel Baccelli, Thomas C. Schmidt, Matthias Wählisch, Cédric Adjih et Laurent Massoulié (sept. 2017). « Low-power Internet of Things with NDN & Cooperative Caching ». Dans : *ACM ICN 2017 - 4th ACM Conference on Information-Centric Networking*. Berlin, Germany.

Harvey, Nicholas J. A., Michael B. Jones, Stefan Saroiu, Marvin Theimer et Alec Wolman (2003). « SkipNet : A Scalable Overlay Network with Practical Locality Properties ». Dans : *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*. USITS'03. Seattle, WA : USENIX Association, p. 9-9.

Heller, Brandon, Rob Sherwood et Nick McKeown (2012). « The Controller Placement Problem ». Dans : *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. HotSDN '12. Helsinki, Finland : ACM, p. 7-12. isbn : 978-1-4503-1477-0. doi : 10.1145/2342441.2342444.

Hertel, Christopher (2003). *Implementing CIFS : The Common Internet File System*. Prentice Hall Professional Technical Reference. isbn : 013047116X.

Higbie, L. C. (déc. 1973). « Tutorial : Supercomputer architecture ». Dans : *Computer* 6.12, p. 48-58. issn : 0018-9162. doi : 10.1109/MC.1973.6540219.

Hildrum, Kirsten, John D. Kubiatowicz, Satish Rao et Ben Y. Zhao (mai 2004). « Distributed Object Location in a Dynamic Network ». Dans : *Theory of Computing Systems* 37.3, p. 405-440. issn : 1433-0490. doi : 10.1007/s00224-004-1146-6.

Hong, Kirak, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder et Boris Koldehofe (2013). « Mobile Fog : A Programming Model for Large-scale Applications on the Internet of Things ». Dans : *Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing*. MCC '13. Hong Kong, China : ACM, p. 15-20. isbn : 978-1-4503-2180-8. doi : 10.1145/2491266.2491270.

Honicky, R. J. et E. L. Miller (avr. 2004). « Replication under scalable hashing : a family of algorithms for scalable decentralized data distribution ». Dans : *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings*. P. 96-. doi : 10.1109/IPDPS.2004.1303042.

Hopkins, S et B Coile (2009). *The ATA over Ethernet Protocol Specification*. Rapp. tech. The Brantley Coile Company, Inc.

Hu, P., H. Ning, T. Qiu, Y. Zhang et X. Luo (août 2017). « Fog Computing Based Face Identification and Resolution Scheme in Internet of Things ». Dans : *IEEE Transactions on Industrial Informatics* 13.4, p. 1910-1920. issn : 1551-3203. doi : 10.1109/TII.2016.2607178.

Huang, Cheng, Angela Wang, Jin Li et Keith W. Ross (2008). « Understanding Hybrid CDN-P2P : Why Limelight Needs Its Own Red Swoosh ». Dans : *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital*

Audio and Video. NOSSDAV '08. Braunschweig, Germany : ACM, p. 75-80. ISBN : 978-1-60558-157-6. DOI : 10.1145/1496046.1496064.

HUANG, Junxian, Feng QIAN, Alexandre GERBER, Z. Morley MAO, Subhabrata SEN et Oliver SPATSCHECK (2012). « A Close Examination of Performance and Power Characteristics of 4G LTE Networks ». Dans : *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*. MobiSys '12. Low Wood Bay, Lake District, UK : ACM, p. 225-238. ISBN : 978-1-4503-1301-8. DOI : 10.1145/2307636.2307658.

HUPFELD, Felix, Toni CORTES, Björn KOLBECK, Jan STENDER, Erich FOCHT, Matthias HESS, Jesus MALO, Jonathan MARTI et Eugenio CESARIO (déc. 2008). « The XtreemFS Architecture ; a Case for Object-based File Systems in Grids ». Dans : *Concurr. Comput. : Pract. Exper.* 20.17, p. 2049-2060. ISSN : 1532-0626. DOI : 10.1002/cpe.v20:17.

IMBERT, Matthieu, Laurent POUILLOUX, Jonathan ROUZAUD-CORNABAS, Adrien LEBRE et Takahiro HIROFUCHI (déc. 2013). « Using the EXECO toolbox to perform automatic and reproducible cloud experiments ». Dans : *1st International Workshop on UsiNg and building ClOud Testbeds (UNICO, collocated with IEEE CloudCom 2013*. Bristol, United Kingdom.

JACOBSON, Van, Diana K. SMETTERS, James D. THORNTON, Michael F. PLASS, Nicholas H. BRIGGS et Rebecca L. BRAYNARD (2009). « Networking Named Content ». Dans : *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*. CoNEXT '09. Rome, Italy : ACM, p. 1-12. ISBN : 978-1-60558-636-6. DOI : 10.1145/1658939.1658941.

JAKAB, L., A. CABELLOS-APARICIO, F. CORAS, D. SAUCEZ et O. BONAVENTURE (oct. 2010). « LISP-TREE : A DNS Hierarchy to Support the LISP Mapping System ». Dans : *IEEE Journal on Selected Areas in Communications* 28.8, p. 1332-1343. ISSN : 0733-8716. DOI : 10.1109/JSAC.2010.101011.

JALALI, F., K. HINTON, R. AYRE, T. ALPCAN et R. S. TUCKER (mai 2016). « Fog Computing May Help to Save Energy in Cloud Computing ». Dans : *IEEE Journal on Selected Areas in Communications* 34.5, p. 1728-1739. ISSN : 0733-8716. DOI : 10.1109/JSAC.2016.2545559.

JARARWEH, Y., A. DOULAT, O. ALQUDAH, E. AHMED, M. AL-AYYOUB et E. BENKHELIFA (mai 2016). « The future of mobile cloud computing : Integrating cloudlets and Mobile Edge Computing ». Dans : *2016 23rd International Conference on Telecommunications (ICT)*, p. 1-5. DOI : 10.1109/ICT.2016.7500486.

JARARWEH, Yaser, Lo'ai TAWALBEH, Fadi ABABNEH, Abdallah KHREISHAH et Fahd DOSARI (2014). « Scalable Cloudlet-based Mobile Computing Model ». Dans : *Procedia Computer Science* 34. The 9th International Conference on Future Networks and Communications (FNC'14)/The 11th International Conference on Mobile Systems

and Pervasive Computing (MobiSPC'14)/Affiliated Workshops, p. 434 -441. ISSN : 1877-0509. DOI : 10.1016/j.procs.2014.07.051.

JIMÉNEZ, Y., C. CERVELLÓ-PASTOR et A. J. GARCÍA (juin 2014). « On the controller placement for designing a distributed SDN control layer ». Dans : *2014 IFIP Networking Conference*, p. 1-9. DOI : 10.1109/IFIPNetworking.2014.6857117.

JUNG, Jaeyeon, E. SIT, H. BALAKRISHNAN et R. MORRIS (oct. 2002). « DNS performance and the effectiveness of caching ». Dans : *IEEE/ACM Transactions on Networking* 10.5, p. 589-603. ISSN : 1063-6692. DOI : 10.1109/TNET.2002.803905.

KAHVAZADEH, Sarang, Vitor B. SOUZA, Xavi MASIP-BRUIN, Eva MARN-TORDERA, Jordi GARCIA et Rodrigo DIAZ (2017). « Securing Combined Fog-to-Cloud System Through SDN Approach ». Dans : *Proceedings of the 4th Workshop on CrossCloud Infrastructures & Platforms*. Crosscloud'17. Belgrade, Serbia : ACM, 2 :1-2 :6. ISBN : 978-1-4503-4934-5. DOI : 10.1145/3069383.3069385.

KANG, H. J., E. CHAN-TIN, N. J. HOPPER et Y. KIM (sept. 2009). « Why Kad lookup fails ». Dans : *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*, p. 121-130. DOI : 10.1109/P2P.2009.5284547.

KANGASHARJU, Jussi, James ROBERTS et Keith W. ROSS (2002). « Object replication strategies in content distribution networks ». Dans : *Computer Communications* 25.4, p. 376 -383. ISSN : 0140-3664. DOI : 10.1016/S0140-3664(01)00409-1.

KARAME, Ghassan (2016). « On the Security and Scalability of Bitcoin's Blockchain ». Dans : *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. Vienna, Austria : ACM, p. 1861-1862. ISBN : 978-1-4503-4139-4. DOI : 10.1145/2976749.2976756.

KARGER, David, Eric LEHMAN, Tom LEIGHTON, Rina PANIGRAHY, Matthew LEVINE et Daniel LEWIN (1997). « Consistent Hashing and Random Trees : Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web ». Dans : *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*. STOC '97. El Paso, Texas, USA : ACM, p. 654-663. ISBN : 0-89791-888-6. DOI : 10.1145/258533.258660.

KATSAK, W., Í. GOIRI, R. BIANCHINI et T. D. NGUYEN (déc. 2015). « GreenCassandra : Using renewable energy in distributed structured storage systems ». Dans : *2015 Sixth International Green and Sustainable Computing Conference (IGSC)*, p. 1-8. DOI : 10.1109/IGCC.2015.7393711.

KAUNE, Sebastian, Tobias LAUINGER, Aleksandra KOVACEVIC et Konstantin PUSSEP (2008). « Embracing the Peer Next Door : Proximity in Kademlia ». Dans : *Proceedings of the 2008 Eighth International Conference on Peer-to-Peer Computing*. P2P '08. Washington, DC, USA : IEEE Computer Society, p. 343-350. ISBN : 978-0-7695-3318-6. DOI : 10.1109/P2P.2008.36.

Kershenbaum, A. et R. Van Slyke (1972). « Computing Minimum Spanning Trees Efficiently ». Dans : *Proceedings of the ACM Annual Conference - Volume 1*. ACM '72. Boston, Massachusetts, USA : ACM, p. 518-527. doi : 10.1145/800193.569966.

Khan, S. U., A. A. Maciejewski et H. J. Siegel (mai 2009). « Robust CDN replica placement techniques ». Dans : *2009 IEEE International Symposium on Parallel Distributed Processing*, p. 1-8. doi : 10.1109/IPDPS.2009.5160908.

Kiesel, Sebastian, Wendy Roome, Richard Woundy, Stefano Previdi, Stanislav Shalunov, Richard Alimi, Reinaldo Penno et Yang Yang (sept. 2014). *Application-Layer Traffic Optimization (ALTO) Protocol*. RFC 7285. doi : 10.17487/RFC7285.

Kim, Jinoh et Eric Chan-tin (2007). *Robust Object Replication in a DHT Ring*.

Kistler, James J. et M. Satyanarayanan (fév. 1992). « Disconnected Operation in the Coda File System ». Dans : *ACM Trans. Comput. Syst.* 10.1, p. 3-25. issn : 0734-2071. doi : 10.1145/146941.146942.

Kitanov, S. et T. Janevski (nov. 2016). « State of the art : Fog computing for 5G networks ». Dans : *2016 24th Telecommunications Forum (TELFOR)*. TELFOR 2016. doi : 10.1109/TELFOR.2016.7818728.

Klemm, A., C. Lindemann et O. P. Waldhorst (oct. 2003). « A special-purpose peer-to-peer file sharing system for mobile ad hoc networks ». Dans : *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall (IEEE Cat. No.03CH37484)*. T. 4, 2758-2763 Vol.4. doi : 10.1109/VETECF.2003.1286080.

Klophaus, Rusty (2010). « Riak Core : Building Distributed Applications Without Shared State ». Dans : *ACM SIGPLAN Commercial Users of Functional Programming*. CUFP '10. Baltimore, Maryland : ACM, 14 :1-14 :1. isbn : 978-1-4503-0516-7. doi : 10.1145/1900160.1900176.

Ko, Bong Jun, Vasileios Pappas, Ramya Raghavendra, Yang Song, Raheleh B. Dilmaghani, Kang-won Lee et Dinesh Verma (2012). « An Information-centric Architecture for Data Center Networks ». Dans : *Proceedings of the Second Edition of the ICN Workshop on Information-centric Networking*. ICN '12. Helsinki, Finland : ACM, p. 79-84. isbn : 978-1-4503-1479-4. doi : 10.1145/2342488.2342506.

Koponen, Teemu, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker et Ion Stoica (août 2007). « A Data-oriented (and Beyond) Network Architecture ». Dans : *SIGCOMM Comput. Commun. Rev.* 37.4, p. 181-192. issn : 0146-4833. doi : 10.1145/1282427.1282402.

Kosar, Tevfik et Miron Livny (2005). « A framework for reliable and efficient data placement in distributed computing systems ». Dans : *Journal of Parallel and Distributed Computing* 65.10. Design and Performance of Networks for Super-, Cluster-, and Grid-Computing Part I, p. 1146 -1157. issn : 0743-7315. doi : 10.1016/j.jpdc.2005.04.019.

Kruskal, Joseph B. (1956). « On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem ». Dans : *Proceedings of the American Mathematical Society* 7.1, p. 48-50. issn : 00029939, 10886826.

Ktari, Salma, Mathieu Zoubert, Artur Hecker et Houda Labiod (2007). « Performance Evaluation of Replication Strategies in DHTs Under Churn ». Dans : *Proceedings of the 6th International Conference on Mobile and Ubiquitous Multimedia*. MUM '07. Oulu, Finland : ACM, p. 90-97. isbn : 978-1-59593-916-6. doi : 10.1145/1329469.1329481.

Kubo, H. et U. C. Kozat (juin 2013). « On improving latency of geographically distributed key-value stores via load balancing with side information ». Dans : *2013 IEEE International Conference on Communications (ICC)*, p. 3710-3715. doi : 10.1109/ICC.2013.6655131.

Kutscher, Dirk, Suyong Eum, Kostas Pentikousis, Ioannis Psaras, Daniel Corujo, Damien Saucez, Thomas C. Schmidt et Matthias Wählisch (juil. 2016). *Information-Centric Networking (ICN) Research Challenges*. RFC 7927. doi : 10.17487/RFC7927.

Lakshman, Avinash et Prashant Malik (avr. 2010). « Cassandra : A Decentralized Structured Storage System ». Dans : *SIGOPS Oper. Syst. Rev.* 44.2, p. 35-40. issn : 0163-5980. doi : 10.1145/1773912.1773922.

Lamport, Leslie (2002). « Paxos Made Simple, Fast, and Byzantine ». Dans : *Procedings of the 6th International Conference on Principles of Distributed Systems. OPODIS 2002, Reims, France, December 11-13, 2002*, p. 7-9.

Lebre, Adrien, Jonathan Pastor, Marin Bertier, Frédéric Desprez, Jonathan Rouzaud-Cornabas, Cédric Tedeschi, Paolo Anedda, Gianluigi Zanetti, Ramon Nou, Toni Cortes, Etienne Rivière et Thomas Ropars (juil. 2013). *Beyond The Cloud, How Should Next Generation Utility Computing Infrastructures Be Designed ?* Research Report RR-8348. INRIA.

Lebre, Adrien et Gustavo Bervian Brand (oct. 2014). « GBFS : Efficient Data-Sharing on Hybrid Platforms. Towards adding WAN-Wide elasticity to DFSes. » Dans : *WPBA Workshop in Proceedings of 26th International Symposium on Computer Architecture and High Performance Computing*. WPBA Workshop in Proceedings of 26th International Symposium on Computer Architecture and High Performance Computing. Paris, France : IEEE.

Legout, Arnaud, Guillaume Urvoy-Keller et Pietro Michiardi (2005). *Understanding BitTorrent : An Experimental Perspective*. Technical Report, p. 16.

Levy, Eliezer et Abraham Silberschatz (déc. 1990). « Distributed File Systems : Concepts and Examples ». Dans : *ACM Comput. Surv.* 22.4, p. 321-374. issn : 0360-0300. doi : 10.1145/98163.98169.

Li, Tonglin, Xiaobing Zhou, Kevin Brandstatter, Dongfang Zhao, Ke Wang, Anupam Rajendran, Zhao Zhang et Ioan Raicu (2013). « ZHT : A Light-Weight Reliable Persistent Dynamic Scalable Zero-Hop Distributed Hash Table ». Dans : *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IPDPS '13. Washington, DC, USA : IEEE Computer Society, p. 775-787. isbn : 978-0-7695-4971-2. doi : 10.1109/IPDPS.2013.110.

Li, Tony (mai 2011). *Design Goals for Scalable Internet Routing*. RFC 6227. doi : 10.17487/RFC6227.

Lin, Guo-Hui et Guoliang Xue (1999). « Steiner tree problem with minimum number of Steiner points and bounded edge-length ». Dans : *Information Processing Letters* 69.2, p. 53 -57. issn : 0020-0190. doi : 10.1016/S0020-0190(98)00201-4.

Lom, M., O. Pribyl et M. Svitek (mai 2016). « Industry 4.0 as a part of smart cities ». Dans : *2016 Smart Cities Symposium Prague (SCSP)*, p. 1-6. doi : 10.1109/SCSP.2016.7501015.

Lu, Yingping et D. H. C. Du (août 2003). « Performance study of iSCSI-based storage subsystems ». Dans : *IEEE Communications Magazine* 41.8, p. 76-82. issn : 0163-6804. doi : 10.1109/MCOM.2003.1222721.

Luan, Tom H, Longxiang Gao, Zhi Li, Yang Xiang, Guiyi Wei et Limin Sun (2015). « Fog computing : Focusing on mobile users at the edge ». Dans : *arXiv preprint arXiv :1502.01815*.

Luciani, Jake (2012). « Cassandra file system design ». Dans :

Ma, Jianwei, Wanyu Liu et Tristan Glatard (août 2013). « A Classification of File Placement and Replication Methods on Grids ». Dans : *Future Gener. Comput. Syst.* 29.6, p. 1395-1406. issn : 0167-739X. doi : 10.1016/j.future.2013.02.006.

Mahmud, Redowan, Ramamohanarao Kotagiri et Rajkumar Buyya (2018). « Fog Computing : A Taxonomy, Survey and Future Directions ». Dans : *Internet of Everything : Algorithms, Methodologies, Technologies and Perspectives*. Sous la dir. de Beniamino Di Martino, Kuan-Ching Li, Laurence T. Yang et Antonio Esposito. Singapore : Springer Singapore, p. 103-130. isbn : 978-981-10-5861-5. doi : 10.1007/978-981-10-5861-5_5.

Malandrino, Francesco, Scott Kirkpatrick et Carla-Fabiana Chiasserini (2016). « How Close to the Edge ? : Delay/Utilization Trends in MEC ». Dans : *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking*. CAN '16. Irvine, California, USA : ACM, p. 37-42. isbn : 978-1-4503-4673-3. doi : 10.1145/3010079.3010080.

Markopoulou, Athina, F. Tobagi et M. Karam (2006). « Loss and Delay Measurements of Internet Backbones ». Dans : *Computer Communications* 29.10. Monitoring and Measurements of IP Networks, p. 1590 -1604. issn : 0140-3664. doi : 10.1016/j.comcom.2005.07.011.

Mascetti, L, E Cano, B Chan, X Espinal, A Fiorot, H González Labrador, J Iven, M Lamanna, G Lo Presti, JT Mościcki, AJ Peters, S Ponce, H Rousseau et D van der Ster (2015). « Disk storage at CERN ». Dans : *Journal of Physics : Conference Series* 664.4, p. 042035.

Masip-Bruin, X., E. Marín-Tordera, A. Alonso et J. Garcia (juin 2016). « Fog-to-cloud Computing (F2C) : The key technology enabler for dependable e-health services deployment ». Dans : *2016 Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, p. 1-5. doi : 10.1109/MedHocNet.2016.7528425.

Mathy, Laurent et Luigi Iannone (2008). « LISP-DHT : Towards a DHT to Map Identifiers Onto Locators ». Dans : *Proceedings of the 2008 ACM CoNEXT Conference*. CoNEXT '08. Madrid, Spain : ACM, 61 :1-61 :6. isbn : 978-1-60558-210-8. doi : 10.1145/1544012.1544073.

Maymounkov, Petar et David Mazières (2002). « Kademlia : A Peer-to-Peer Information System Based on the XOR Metric ». Dans : *Revised Papers from the First International Workshop on Peer-to-Peer Systems*. IPTPS '01. London, UK, UK : Springer-Verlag, p. 53-65. isbn : 3-540-44179-4.

Melman, David T., Tal Mizrahi et Donald E. Eastlake 3rd (jan. 2013). *Fibre Channel over Ethernet (FCoE) over Transparent Interconnection of Lots of Links (TRILL)*. RFC 6847. doi : 10.17487/RFC6847.

Mockapetris, P. (nov. 1987). *Domain Names - Concepts and Facilities*. RFC 1034. Network Working Group. doi : 10.17487/RFC1034.

Morin, Christine, Pascal Gallard, Renaud Lottiaux et Geoffroy Vallée (mai 2004). « Towards an Efficient Single System Image Cluster Operating System ». Dans : *Future Gener. Comput. Syst.* 20.4, p. 505-521. issn : 0167-739X. doi : 10.1016/S0167-739X(03)00170-5.

Muralidhar, Subramanian, Wyatt Lloyd, Sabyasachi Roy, Cory Hill, Ernest Lin, Weiwen Liu, Satadru Pan, Shiva Shankar, Viswanath Sivakumar, Linpeng Tang et Sanjeev Kumar (2014). « F4 : Facebook's Warm BLOB Storage System ». Dans : *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*. OSDI'14. Broomfield, CO : USENIX Association, p. 383-398. isbn : 978-1-931971-16-4.

Muthitacharoen, Athicha, Robert Morris, Thomer M. Gil et Benjie Chen (déc. 2002). « Ivy : A Read/Write Peer-to-peer File System ». Dans : *SIGOPS Oper. Syst. Rev.* 36.SI, p. 31-44. issn : 0163-5980. doi : 10.1145/844128.844132.

Nakamoto, Satoshi (2008). *Bitcoin : A Peer-to-Peer Electronic Cash System*. Rapp. tech. Nakamoto Institute.

Niazi, Salman, Mahmoud Ismail, Seif Haridi, Jim Dowling, Steffen Grohsschmiedt et Mikael Ronström (2017). « HopsFS : Scaling Hierarchical File System Metadata

Using newSQL Databases ». Dans : *Proceedings of the 15th Usenix Conference on File and Storage Technologies*. FAST'17. Santa clara, CA, USA : USENIX Association, p. 89-103. ISBN : 978-1-931971-36-2.

NIELSEN, Jakob (1998). « Nielsen's law of internet bandwidth ». Dans :

NIGHTINGALE, Edmund B., Jeremy ELSON, Jinliang FAN, Owen HOFMANN, Jon HOWELL et Yutaka SUZUE (2012). « Flat Datacenter Storage ». Dans : *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*. OSDI'12. Hollywood, CA, USA : USENIX Association, p. 1-15. ISBN : 978-1-931971-96-6.

NORDRUM, Amy (2016). « Popular internet of things forecast of 50 billion devices by 2020 is outdated ». Dans : *IEEE Spectrum* 18.

NYGREN, Erik, Ramesh K. SITARAMAN et Jennifer SUN (août 2010). « The Akamai Network : A Platform for High-performance Internet Applications ». Dans : *SIGOPS Oper. Syst. Rev.* 44.3, p. 2-19. ISSN : 0163-5980. DOI : 10.1145/1842733.1842736.

OH, Kwangsung, Ajaykrishna RAGHAVAN, Abhishek CHANDRA et Jon WEISSMAN (2015). « Redefining Data Locality for Cross-Data Center Storage ». Dans : *Proceedings of the 2Nd International Workshop on Software-Defined Ecosystems*. BigSystem '15. Portland, Oregon, USA : ACM, p. 15-22. ISBN : 978-1-4503-3568-3. DOI : 10.1145/2756594.2756596.

OLANIYAN, Richard et Muthucumaru MAHESWARAN (mai 2018). « Synchronous Scheduling Algorithms for Edge Coordinated Internet of Things ». Dans : *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*.

PACHECO, L., R. HALALAI, V. SCHIAVONI, F. PEDONE, E. RIVIÈRE et P. FELBER (sept. 2016). « GlobalFS : A Strongly Consistent Multi-site File System ». Dans : *2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*, p. 147-156. DOI : 10.1109/SRDS.2016.027.

PADHYE, Jitendra, Victor FIROIU, Don TOWSLEY et Jim KUROSE (oct. 1998). « Modeling TCP Throughput : A Simple Model and Its Empirical Validation ». Dans : *SIGCOMM Comput. Commun. Rev.* 28.4, p. 303-314. ISSN : 0146-4833. DOI : 10.1145/285243.285291.

PAIVA, João et Luís RODRIGUES (2013). « Policies for Efficient Data Replication in P2P Systems ». Dans : *Proceedings of the 2013 International Conference on Parallel and Distributed Systems*. ICPADS '13. Washington, DC, USA : IEEE Computer Society, p. 404-411. ISBN : 978-1-4799-2081-5. DOI : 10.1109/.62.

PAIVA, João, Pedro RUIVO, Paolo ROMANO et Luís RODRIGUES (déc. 2014). « AutoPlacer : Scalable Self-Tuning Data Placement in Distributed Key-Value Stores ». Dans : *ACM Transactions on Autonomous and Adaptive Systems* 9.4, 19 :1-19 :30. ISSN : 1556-4665. DOI : 10.1145/2641573.

Paiva, João et Luís Rodrigues (jan. 2015). « On Data Placement in Distributed Systems ». Dans : *SIGOPS Oper. Syst. Rev.* 49.1, p. 126-130. issn : 0163-5980. doi : 10.1145/2723872.2723890.

Palazzi, C. E., A. Bujari et E. Cervi (août 2009). « P2P file sharing on mobile phones : Design and implementation of a prototype ». Dans : *2009 2nd IEEE International Conference on Computer Science and Information Technology*, p. 136-140. doi : 10.1109/ICCSIT.2009.5234836.

Pang, Z., L. Sun, Z. Wang, E. Tian et S. Yang (nov. 2015). « A Survey of Cloudlet Based Mobile Computing ». Dans : *2015 International Conference on Cloud Computing and Big Data (CCBD)*, p. 268-275. doi : 10.1109/CCBD.2015.54.

Pappas, V., D. Massey, A. Terzis et L. Zhang (avr. 2006). « A Comparative Study of the DNS Design with DHT-Based Alternatives ». Dans : *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, p. 1-13. doi : 10.1109/INFOCOM.2006.207.

Park, KyoungSoo, Vivek S. Pai, Larry Peterson et Zhe Wang (2004). « CoDNS : Improving DNS Performance and Reliability via Cooperative Lookups ». Dans : *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*. OSDI'04. San Francisco, CA : USENIX Association, p. 14-14.

Passarella, Andrea (2012). « A survey on content-centric technologies for the current Internet : CDN and P2P solutions ». Dans : *Computer Communications* 35.1, p. 1 -32. issn : 0140-3664. doi : 10.1016/j.comcom.2011.10.005.

Pertin, Dimitri, Sylvain David, Pierre Évenou, Benoît Parrein et Nicolas Normand (avr. 2014). « Distributed File System based on Erasure Coding for I/O Intensive Applications ». Dans : *4th International Conference on Cloud Computing and Service Science*. Barcelone, Spain.

Ping, Fan, Jeong-Hyon Hwang, XiaoHu Li, Chris McConnell et Rohini Vabbalareddy (2011). « Wide Area Placement of Data Replicas for Fast and Highly Available Data Access ». Dans : *Proceedings of the Fourth International Workshop on Data-intensive Distributed Computing*. DIDC '11. San Jose, California, USA : ACM, p. 1-8. isbn : 978-1-4503-0704-8. doi : 10.1145/1996014.1996016.

Plaxton, C. G. et R. Rajaraman (oct. 1996). « Fast fault-tolerant concurrent access to shared objects ». Dans : *Proceedings of 37th Conference on Foundations of Computer Science*, p. 570-579. doi : 10.1109/SFCS.1996.548516.

Plaxton, C. G., R. Rajaraman et A. W. Richa (juin 1999). « Accessing Nearby Copies of Replicated Objects in a Distributed Environment ». Dans : *Theory of Computing Systems* 32.3, p. 241-280. issn : 1433-0490. doi : 10.1007/s002240000118.

Popoviciu, Chip, Christian Hahn, Olaf Bonness, Gunter Van de Velde et Tim Chown (déc. 2008). *IPv6 Unicast Address Assignment Considerations*. RFC 5375. doi : 10.17487/RFC5375.

Prim, R. C. (nov. 1957). « Shortest connection networks and some generalizations ». Dans : *The Bell System Technical Journal* 36.6, p. 1389-1401. issn : 0005-8580. doi : 10.1002/j.1538-7305.1957.tb01515.x.

Qin, Z., G. Denker, C. Giannelli, P. Bellavista et N. Venkatasubramanian (mai 2014). « A Software Defined Networking architecture for the Internet-of-Things ». Dans : *2014 IEEE Network Operations and Management Symposium (NOMS)*, p. 1-9. doi : 10.1109/NOMS.2014.6838365.

Qiu, Lili, V. N. Padmanabhan et G. M. Voelker (2001). « On the placement of Web server replicas ». Dans : *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*. T. 3, 1587-1596 vol.3. doi : 10.1109/INFCOM.2001.916655.

Rahman, A., W. Olesinski et P. Gburzynski (mai 2004). « Controlled flooding in wireless ad-hoc networks ». Dans : *International Workshop on Wireless Ad-Hoc Networks, 2004.* P. 73-78. doi : 10.1109/IWWAN.2004.1525544.

Ramasubramanian, Venugopalan et Emin Gün Sirer (août 2004). « The Design and Implementation of a Next Generation Name Service for the Internet ». Dans : *SIGCOMM Comput. Commun. Rev.* 34.4, p. 331-342. issn : 0146-4833. doi : 10.1145/1030194.1015504.

Ratnasamy, Sylvia, Paul Francis, Mark Handley, Richard Karp et Scott Shenker (août 2001). « A Scalable Content-addressable Network ». Dans : *SIGCOMM Comput. Commun. Rev.* 31.4, p. 161-172. issn : 0146-4833. doi : 10.1145/964723.383072.

Rhea, Sean, Dennis Geels, Timothy Roscoe et John Kubiatowicz (2004). « Handling Churn in a DHT ». Dans : *Proceedings of the Annual Conference on USENIX Annual Technical Conference*. ATEC '04. Boston, MA : USENIX Association, p. 10-10.

Rose, Scott, Matt Larson, Dan Massey, Rob Austein et Roy Arends (mar. 2005). *DNS Security Introduction and Requirements*. RFC 4033. doi : 10.17487/RFC4033.

Rowstron, Antony et Peter Druschel (2001). « Pastry : Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems ». Dans : *Middleware 2001*. Sous la dir. de Rachid Guerraoui. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 329-350. isbn : 978-3-540-45518-9.

Samaniego, M. et R. Deters (déc. 2016). « Blockchain as a Service for IoT ». Dans : *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and*

Social Computing (CPSCom) and IEEE Smart Data (SmartData), p. 433-436. DOI : 10.1109/iThings-GreenCom-CPSCom-SmartData.2016.102.

SATHIAMOORTHY, Maheswaran, Megasthenis ASTERIS, Dimitris PAPAILIOPOULOS, Alexandros G. DIMAKIS, Ramkumar VADALI, Scott CHEN et Dhruba BORTHAKUR (mar. 2013). « XORing Elephants : Novel Erasure Codes for Big Data ». Dans : *Proc. VLDB Endow.* 6.5, p. 325-336. ISSN : 2150-8097. DOI : 10.14778/2535573.2488339.

SATYANARAYANAN, M., P. SIMOENS, Y. XIAO, P. PILLAI, Z. CHEN, K. HA, W. HU et B. AMOS (avr. 2015). « 0Edge Analytics in the Internet of Things ». Dans : *IEEE Pervasive Computing* 14.2, p. 24-31. ISSN : 1536-1268. DOI : 10.1109/MPRV.2015.32.

SATYANARAYANAN, Mahadev (1990). « A survey of distributed file systems ». Dans : *Annual Review of Computer Science* 4.1, p. 73-104.

SATYANARAYANAN, Mahadev (2013). « Cloudlets : At the Leading Edge of Cloud-mobile Convergence ». Dans : *Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures*. QoSA '13. Vancouver, British Columbia, Canada : ACM, p. 1-2. ISBN : 978-1-4503-2126-6. DOI : 10.1145/2465478.2465494.

SAUREZ, Enrique, Kirak HONG, Dave LILLETHUN, Umakishore RAMACHANDRAN et Beate OTTENWÄLDER (2016). « Incremental Deployment and Migration of Geo-distributed Situation Awareness Applications in the Fog ». Dans : *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*. DEBS '16. Irvine, California : ACM, p. 258-269. ISBN : 978-1-4503-4021-2. DOI : 10.1145/2933267.2933317.

SELIMI, M. et F. FREITAG (déc. 2014). « Tahoe-LAFS Distributed Storage Service in Community Network Clouds ». Dans : *2014 IEEE Fourth International Conference on Big Data and Cloud Computing*, p. 17-24. DOI : 10.1109/BDCloud.2014.24.

SERBU, Sabina, Pascal FELBER et Peter KROPF (2011). « HyPeer : Structured overlay with flexible-choice routing ». Dans : *Computer Networks* 55.1, p. 300 -313. ISSN : 1389-1286. DOI : 10.1016/j.comnet.2010.09.006.

SEVILLA, Michael A., Noah WATKINS, Carlos MALTZAHN, Ike NASSI, Scott A. BRANDT, Sage A. WEIL, Greg FARNUM et Sam FINEBERG (2015). « Mantle : A Programmable Metadata Load Balancer for the Ceph File System ». Dans : *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '15. Austin, Texas : ACM, 21 :1-21 :12. ISBN : 978-1-4503-3723-6. DOI : 10.1145/2807591.2807607.

SHAH, R. C., S. WIETHOLTER, A. WOLISZ et J. M. RABAEY (mar. 2005a). « When does opportunistic routing make sense ? » Dans : *Third IEEE International Conference on Pervasive Computing and Communications Workshops*, p. 350-356. DOI : 10.1109/PERCOMW.2005.90.

Shah, Rahul C., Sven Wietholter, Adam Wolisz et Jan M. Rabaey (2005b). « When Does Opportunistic Routing Make Sense ? » Dans : *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*. PERCOMW '05. Washington, DC, USA : IEEE Computer Society, p. 350-356. isbn : 0-7695-2300-5. doi : 10.1109/PERCOMW.2005.90.

Sharma, Prateek, Tian Guo, Xin He, David Irwin et Prashant Shenoy (2016). « Flint : Batch-interactive Data-intensive Processing on Transient Servers ». Dans : *Proceedings of the Eleventh European Conference on Computer Systems*. EuroSys '16. London, United Kingdom : ACM, 6 :1-6 :15. isbn : 978-1-4503-4240-7. doi : 10.1145/2901318.2901319.

Shen, Ying (2014). *Global Internet routing table reaches 512k milestone*. Rapp. tech. Cisco.

Shi, W., J. Cao, Q. Zhang, Y. Li et L. Xu (oct. 2016). « Edge Computing : Vision and Challenges ». Dans : *IEEE Internet of Things Journal* 3.5, p. 637-646. issn : 2327-4662. doi : 10.1109/JIOT.2016.2579198.

Shvachko, K., H. Kuang, S. Radia et R. Chansler (mai 2010a). « The Hadoop Distributed File System ». Dans : *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, p. 1-10. doi : 10.1109/MSST.2010.5496972.

Shvachko, Konstantin, Hairong Kuang, Sanjay Radia et Robert Chansler (2010b). « The Hadoop Distributed File System ». Dans : *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. MSST '10. Washington, DC, USA : IEEE Computer Society, p. 1-10. isbn : 978-1-4244-7152-2. doi : 10.1109/MSST.2010.5496972.

Simms, Stephen C., Gregory G. Pike et Doug Balog (2007). « Wide Area Filesystem Performance using Lustre on the TeraGrid ». Dans : *in Proceedings of the TeraGrid 2007 Conference*.

Siris, Vasilios A. et Dimitrios Kalyvas (2012). « Enhancing Mobile Data Offloading with Mobility Prediction and Prefetching ». Dans : *Proceedings of the Seventh ACM International Workshop on Mobility in the Evolving Internet Architecture*. MobiArch '12. Istanbul, Turkey : ACM, p. 17-22. isbn : 978-1-4503-1526-5. doi : 10.1145/2348676.2348682.

Sit, Emil, Frank Dabek et James Robertson (2005). « UsenetDHT : A Low Overhead Usenet Server ». Dans : *Peer-to-Peer Systems III*. Sous la dir. de Geoffrey M. Voelker et Scott Shenker. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 206-216. isbn : 978-3-540-30183-7.

Sivasubramanian, Swaminathan, Gustavo Alonso, Guillaume Pierre et Maarten van Steen (2005). « GlobeDB : Autonomic Data Replication for Web Applications ». Dans : *Proceedings of the 14th International Conference on World Wide Web*. WWW

'05. Chiba, Japan : ACM, p. 33-42. ISBN : 1-59593-046-9. DOI : 10.1145/1060745.1060756.

Souza Couto, R. D., S. Secci, M. E. Mitre Campista et L. H. Maciel Kosmalski Costa (oct. 2014). « Network design requirements for disaster resilience in IaaS Clouds ». Dans : *IEEE Communications Magazine* 52.10, p. 52-58. ISSN : 0163-6804. DOI : 10.1109/MCOM.2014.6917402.

Stoica, Ion, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek et Hari Balakrishnan (fév. 2003). « Chord : A Scalable Peer-to-peer Lookup Protocol for Internet Applications ». Dans : *IEEE/ACM Trans. Netw.* 11.1, p. 17-32. ISSN : 1063-6692. DOI : 10.1109/TNET.2002.808407.

Stojmenovic, I. et S. Wen (sept. 2014). « The Fog computing paradigm : Scenarios and security issues ». Dans : *2014 Federated Conference on Computer Science and Information Systems*, p. 1-8. DOI : 10.15439/2014F503.

Sui, Kaixin, Mengyu Zhou, Dapeng Liu, Minghua Ma, Dan Pei, Youjian Zhao, Zimu Li et Thomas Moscibroda (2016). « Characterizing and Improving WiFi Latency in Large-Scale Operational Networks ». Dans : *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '16. Singapore, Singapore : ACM, p. 347-360. ISBN : 978-1-4503-4269-8. DOI : 10.1145/2906388.2906393.

Sun, G., G. Liu, H. Zhang et W. Tan (nov. 2013). « Architecture on mobility management in OpenFlow-based radio access networks ». Dans : *2013 IEEE Global High Tech Congress on Electronics*, p. 88-92. DOI : 10.1109/GHTCE.2013.6767247.

Tang, Bo, Zhen Chen, Gerald Hefferman, Tao Wei, Haibo He et Qing Yang (2015). « A Hierarchical Distributed Fog Computing Architecture for Big Data Analysis in Smart Cities ». Dans : *Proceedings of the ASE BigData &#38 ; SocialInformatics 2015*. ASE BD&#38 ;SI '15. Kaohsiung, Taiwan : ACM, 28 :1-28 :6. ISBN : 978-1-4503-3735-9. DOI : 10.1145/2818869.2818898.

Tatebe, Osamu, Kohei Hiraga et Noriyuki Soda (juil. 2010). « Gfarm Grid File System ». Dans : *New Generation Computing* 28.3, p. 257-275. ISSN : 1882-7055. DOI : 10.1007/s00354-009-0089-5.

Tirado, J. M., D. Higuero, F. Isaila et J. Carretero (mai 2011). « Predictive Data Grouping and Placement for Cloud-Based Elastic Server Infrastructures ». Dans : *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, p. 285-294. DOI : 10.1109/CCGrid.2011.49.

Tran, Minh et Wallapak Tavanapong (2005). « On Using a CDN's Infrastructure to Improve File Transfer Among Peers ». Dans : *Management of Multimedia Networks and Services*. Sous la dir. de Jordi Dalmau Royo et Go Hasegawa. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 289-301. ISBN : 978-3-540-32090-6.

Tran, Nguyen, Marcos K. Aguilera et Mahesh Balakrishnan (2011). « Online Migration for Geo-distributed Storage Systems ». Dans : *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*. USENIXATC'11. Portland, OR : USENIX Association, p. 15-15.

Tran, T. X., A. Hajisami et D. Pompili (juil. 2017). « Cooperative Hierarchical Caching in 5G Cloud Radio Access Networks ». Dans : *IEEE Network* 31.4, p. 35-41. issn : 0890-8044. doi : 10.1109/MNET.2017.1600307.

Truong, N. B., G. M. Lee et Y. Ghamri-Doudane (mai 2015). « Software defined networking-based vehicular Adhoc Network with Fog Computing ». Dans : *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, p. 1202-1207. doi : 10.1109/INM.2015.7140467.

Vaquero, Luis M. et Luis Rodero-Merino (oct. 2014). « Finding Your Way in the Fog : Towards a Comprehensive Definition of Fog Computing ». Dans : *SIGCOMM Comput. Commun. Rev.* 44.5, p. 27-32. issn : 0146-4833. doi : 10.1145/2677046.2677052.

Varshney, P. et Y. Simmhan (mai 2017). « Demystifying Fog Computing : Characterizing Architectures, Applications and Abstractions ». Dans : *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, p. 115-124. doi : 10.1109/ICFEC.2017.20.

Vasilakos, Athanasios V., Zhe Li, Gwendal Simon et Wei You (2015). « Information centric network : Research challenges and opportunities ». Dans : *Journal of Network and Computer Applications* 52, p. 1 -10. issn : 1084-8045. doi : 10.1016/j.jnca.2015.02.001.

Veltri, L., G. Morabito, S. Salsano, N. Blefari-Melazzi et A. Detti (juin 2012). « Supporting information-centric functionality in software defined networks ». Dans : *2012 IEEE International Conference on Communications (ICC)*, p. 6645-6650. doi : 10.1109/ICC.2012.6364916.

Vilalta, R., A. Mayoral, R. Casellas, R. Martínez et R. Muñoz (juil. 2016). « SDN/NFV orchestration of multi-technology and multi-domain networks in cloud/fog architectures for 5g services ». Dans : *2016 21st OptoElectronics and Communications Conference (OECC) held jointly with 2016 International Conference on Photonics in Switching (PS)*, p. 1-3.

Vixie, Paul (1995). « External Issues in DNS Scalability ». Dans : *Harvard Connection*.

Vorick, David et Luke Champine (2014). *Sia : Simple Decentralized Storage*. Rapp. tech. NebulousLabs, Boston.

Wah, Benjamin W (1986). « Distributed Systems, Vol. II : Distributed Data Base Systems ». Dans : sous la dir. de Wesley W Chu. Norwood, MA, USA : Artech House, Inc. Chap. File Placement on Distributed Computer Systems, p. 19-28. isbn : 0-89006-213-7.

Walgenbach, Joshua, Stephen C. Simms, Kit Westneat et Justin P. Miller (2010). « Enabling Lustre WAN for Production Use on the TeraGrid : A Lightweight UID Mapping Scheme ». Dans : *Proceedings of the 2010 TeraGrid Conference*. TG '10. Pittsburgh, Pennsylvania : ACM, 19 :1-19 :6. isbn : 978-1-60558-818-6. doi : 10.1145/1838574.1838593.

Wan, J., S. Tang, Z. Shu, D. Li, S. Wang, M. Imran et A. V. Vasilakos (oct. 2016). « Software-Defined Industrial Internet of Things in the Context of Industry 4.0 ». Dans : *IEEE Sensors Journal* 16.20, p. 7373-7380. issn : 1530-437X. doi : 10.1109/JSEN.2016.2565621.

Weil, Sage A., Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long et Carlos Maltzahn (2006a). « Ceph : A Scalable, High-performance Distributed File System ». Dans : *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*. OSDI '06. Seattle, Washington : USENIX Association, p. 307-320. isbn : 1-931971-47-1.

Weil, Sage A., Scott A. Brandt, Ethan L. Miller et Carlos Maltzahn (2006b). « CRUSH : Controlled, Scalable, Decentralized Placement of Replicated Data ». Dans : *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*. SC '06. Tampa, Florida : ACM. isbn : 0-7695-2700-0. doi : 10.1145/1188455.1188582.

Weil, Sage A., Andrew W. Leung, Scott A. Brandt et Carlos Maltzahn (2007). « RADOS : A Scalable, Reliable Storage Service for Petabyte-scale Storage Clusters ». Dans : *Proceedings of the 2Nd International Workshop on Petascale Data Storage : Held in Conjunction with Supercomputing '07*. PDSW '07. Reno, Nevada : ACM, p. 35-44. isbn : 978-1-59593-899-2. doi : 10.1145/1374596.1374606.

White, Tom (2009). *Hadoop : The Definitive Guide*. 1st. O'Reilly Media, Inc. isbn : 0596521979, 9780596521974.

Wilkinson, Shawn, Tome Boshevski, Josh Brandoff et Vitalik Buterin (2014). *Storj A Peer-to-Peer Cloud Storage Network*. Rapp. tech. Storj Labs Inc.

Wires, Jake et Andrew Warfield (2017). « Mirador : An Active Control Plane for Datacenter Storage ». Dans : *Proceedings of the 15th Usenix Conference on File and Storage Technologies*. FAST'17. Santa clara, CA, USA : USENIX Association, p. 213-227. isbn : 978-1-931971-36-2.

Wood, Gavin et Vitalik Buterin (2013). *Ethereum white paper*. Rapp. tech. Ethcore inc.

Wu, W., Y. Chen, X. Zhang, X. Shi, L. Cong, B. Deng et X. Li (jan. 2008). « LDHT : Locality-aware Distributed Hash Tables ». Dans : *2008 International Conference on Information Networking*, p. 1-5. doi : 10.1109/ICOIN.2008.4472811.

Wu, Zhe, Michael Butkiewicz, Dorian Perkins, Ethan Katz-Bassett et Harsha V. Madhyastha (2013). « SPANStore : Cost-effective Geo-replicated Storage Spanning

Multiple Cloud Services ». Dans : *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. SOSP '13. Farminton, Pennsylvania : ACM, p. 292-308. ISBN : 978-1-4503-2388-8. DOI : 10.1145/2517349.2522730.

XIE, Haiyong, Y. Richard YANG, Arvind KRISHNAMURTHY, Yanbin Grace LIU et Abraham SILBERSCHATZ (août 2008). « P4P : Provider Portal for Applications ». Dans : *SIGCOMM Comput. Commun. Rev.* 38.4, p. 351-362. ISSN : 0146-4833. DOI : 10.1145/1402946.1402999.

XU, Dongyan, Sunil Suresh KULKARNI, Catherine ROSENBERG et Heung-Keung CHAI (avr. 2006). « Analysis of a CDN–P2P hybrid architecture for cost-effective streaming media distribution ». Dans : *Multimedia Systems* 11.4, p. 383-399. ISSN : 1432-1882. DOI : 10.1007/s00530-006-0015-3.

XU, L. D., W. HE et S. LI (nov. 2014). « Internet of Things in Industries : A Survey ». Dans : *IEEE Transactions on Industrial Informatics* 10.4, p. 2233-2243. ISSN : 1551-3203. DOI : 10.1109/TII.2014.2300753.

YANG, Zhi, Ben Y. ZHAO, Yuanjian XING, Song DING, Feng XIAO et Yafei DAI (2010). « AmazingStore : Available, Low-cost Online Storage Service Using Cloudlets ». Dans : *Proceedings of the 9th International Conference on Peer-to-peer Systems*. IPTPS'10. San Jose, CA : USENIX Association, p. 2-2.

YANNUZZI, M., R. MILITO, R. SERRAL-GRACIÀ, D. MONTERO et M. NEMIROVSKY (déc. 2014). « Key ingredients in an IoT recipe : Fog Computing, Cloud computing, and more Fog Computing ». Dans : *2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, p. 325-329. DOI : 10.1109/CAMAD.2014.7033259.

YEGANEH, S. H., A. TOOTOONCHIAN et Y. GANJALI (fév. 2013). « On scalability of software-defined networking ». Dans : *IEEE Communications Magazine* 51.2, p. 136-141. ISSN : 0163-6804. DOI : 10.1109/MCOM.2013.6461198.

YI, S., Z. HAO, Z. QIN et Q. LI (nov. 2015a). « Fog Computing : Platform and Applications ». Dans : *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, p. 73-78. DOI : 10.1109/HotWeb.2015.22.

YI, Shanhe, Cheng LI et Qun LI (2015b). « A Survey of Fog Computing : Concepts, Applications and Issues ». Dans : *Proceedings of the 2015 Workshop on Mobile Big Data*. Mobidata '15. Hangzhou, China : ACM, p. 37-42. ISBN : 978-1-4503-3524-9. DOI : 10.1145/2757384.2757397.

YI, Shanhe, Zhengrui QIN et Qun LI (2015c). « Security and Privacy Issues of Fog Computing : A Survey ». Dans : *Wireless Algorithms, Systems, and Applications*. Sous la dir. de Kuai XU et Haojin ZHU. Cham : Springer International Publishing, p. 685-695. ISBN : 978-3-319-21837-3.

Yousefpour, A., G. Ishigaki et J. P. Jue (juin 2017). « Fog Computing : Towards Minimizing Delay in the Internet of Things ». Dans : *2017 IEEE International Conference on Edge Computing (EDGE)*, p. 17-24. doi : 10.1109/IEEE.EDGE.2017.12.

Zaharia, M. et S. Keshav (fév. 2008). « Gossip-based Search Selection in Hybrid Peer-to-peer Networks : Research Articles ». Dans : *Concurr. Comput. : Pract. Exper.* 20.2, p. 139-153. issn : 1532-0626. doi : 10.1002/cpe.v20:2.

Zaharia, Matei, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker et Ion Stoica (2010). « Spark : Cluster Computing with Working Sets ». Dans : *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*. HotCloud'10. Boston, MA : USENIX Association, p. 10-10.

Zahn, Thomas et Jochen Schiller (2005). « MADPastry : A DHT Substrate for Practicably Sized MANETs ». Dans : *in Proc. of 5th Workshop on Applications and Services in Wireless Networks (ASWN2005)*.

Zanella, A., N. Bui, A. Castellani, L. Vangelista et M. Zorzi (fév. 2014). « Internet of Things for Smart Cities ». Dans : *IEEE Internet of Things Journal* 1.1, p. 22-32. issn : 2327-4662. doi : 10.1109/JIOT.2014.2306328.

Zeidner, Efri, Constantine Sapuntzakis, Mallikarjun Chadalapaka, Julian Satran et Kalman Meth (avr. 2004). *Internet Small Computer Systems Interface (iSCSI)*. RFC 3720. doi : 10.17487/RFC3720.

Zeydan, E., E. Bastug, M. Bennis, M. A. Kader, I. A. Karatepe, A. S. Er et M. Debbah (sept. 2016). « Big data caching for networking : moving from cloud to edge ». Dans : *IEEE Communications Magazine* 54.9, p. 36-42. issn : 0163-6804. doi : 10.1109/MCOM.2016.7565185.

Zhang, Ben, Nitesh Mor, John Kolb, Douglas S. Chan, Nikhil Goyal, Ken Lutz, Eric Allman, John Wawrzynek, Edward Lee et John Kubiatowicz (2015). « The Cloud is Not Enough : Saving Iot from the Cloud ». Dans : *Proceedings of the 7th USENIX Conference on Hot Topics in Cloud Computing*. HotCloud'15. Santa Clara, CA : USENIX Association, p. 21-21.

Zhao, B. Y., Ling Huang, J. Stribling, S. C. Rhea, A. D. Joseph et J. D. Kubiatowicz (sept. 2006). « Tapestry : A Resilient Global-scale Overlay for Service Deployment ». Dans : *IEEE J.Sel. A. Commun.* 22.1, p. 41-53. issn : 0733-8716. doi : 10.1109/JSAC.2003.818784.

Zhao, Ben Y., John D. Kubiatowicz et Anthony D. Joseph (jan. 2002). « Tapestry : A Fault-tolerant Wide-area Application Infrastructure ». Dans : *SIGCOMM Comput. Commun. Rev.* 32.1, p. 81-81. issn : 0146-4833. doi : 10.1145/510726.510755.

Zhu, J., D. S. Chan, M. S. Prabhu, P. Natarajan, H. Hu et F. Bonomi (mar. 2013). « Improving Web Sites Performance Using Edge Servers in Fog Computing

Architecture ». Dans : *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, p. 320-323. DOI : 10.1109/SOSE.2013.73.

ZHU, Y. et Y. HU (avr. 2005). « Efficient, proximity-aware load balancing for DHT-based P2P systems ». Dans : *IEEE Transactions on Parallel and Distributed Systems* 16.4, p. 349-361. ISSN : 1045-9219. DOI : 10.1109/TPDS.2005.46.

ZOHAR, Aviv (août 2015). « Bitcoin : Under the Hood ». Dans : *Commun. ACM* 58.9, p. 104-113. ISSN : 0001-0782. DOI : 10.1145/2701411.

Production scientifique

6.5 Journaux

Confais, Bastien, Adrien Lebre et Benoît Parrein (août 2017e). « Performance Analysis of Object Store Systems in a Fog and Edge Computing Infrastructure ». Dans : *Transactions on Large-Scale Data- and Knowledge-Centered Systems XXXIII*. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 40-79. isbn : 978-3-662-55696-2. doi : 10.1007/978-3-662-55696-2_2.

6.6 Conférences internationales avec comité de lecture

Confais, Bastien, Adrien Lebre et Benoît Parrein (déc. 2016b). « Performance Analysis of Object Store Systems in a Fog/Edge Computing Infrastructures ». Dans : *CloudCom*. Luxembourg, Luxembourg, Décembre 2016.

Confais, Bastien, Adrien Lebre et Benoît Parrein (mai 2017c). « An Object Store Service for a Fog/Edge Computing Infrastructure based on IPFS and Scale-out NAS ». Dans : *1st IEEE International Conference on Fog and Edge Computing - ICFEC'2017*. Madrid, Spain, Mai 2017.

Confais, Bastien, Adrien Lebre et Benoît Parrein (2018b). « A Tree-Based Approach to locate Object Replicas in a Fog Storage Infrastructure (en soumission) ». Dans : *IEEE Global Communications Conference (GlobeCom 2018)*.

6.7 Conférences nationales avec comité de lecture

Confais, Bastien, Adrien Lebre et Benoît Parrein (juil. 2016d). « Quel système de stockage pour les architectures Fog ? » Dans : *Compas'2016*. Lorient, France.

Confais, Bastien, Adrien Lebre et Benoît Parrein (juin 2017a). « An object store for Fog infrastructures based on IPFS and a Scale-Out NAS ». Dans : *RESCOM 2017*. RESCOM 2017 École d'été, journées thématiques (virtualisation dans les réseaux et le Cloud). Le Croisic, France, 2, (session poster).

Confais, Bastien, Adrien Lebre et Benoît Parrein (avr. 2018f). « Improving locality of an object store working in a Fog environment ». Dans : *1st Grid'5000-FIT school*. Nice, France.

6.8 Autres publications

Confais, Bastien (avr. 2017). « Un système de stockage par objets pour les architectures Fog s'appuyant sur IPFS et un système de fichiers distribué de type Scale-Out NAS ». Dans : *Journée des doctorants 2017, ED STIM*. Nantes, France.

Confais, Bastien, Adrien Lebre et Benoît Parrein (juin 2016c). « Quel système de stockage distribué pour le Fog ? » Dans : *Journées Scientifiques de l'Université de Nantes*. Nantes, France.

Confais, Bastien, Adrien Lebre et Benoît Parrein (nov. 2016e). « Which storage system for FOG Computing ? (conférence invitée) ». Dans : *WOS6 : 6th INRIA/Technicolor workshop on Big Data and Analytics*. Rennes, France.

Confais, Bastien, Adrien Lebre et Benoît Parrein (jan. 2018d). « Adaptation of the Dijkstra's algorithm for metadata management in Fog Computing ». Dans : *Journées non thématiques, RESCOM 2018*. Toulouse, France.

Production logicielle

- Module pour *Yahoo Cloud System Benchmark* (YCSB) afin d'évaluer les performances d'*InterPlantery FileSystem* (IPFS), 160 lignes en Java, https://github.com/bconfais/YCSB/blob/master/ipfs/src/main/java/com/yahoo/ycsb/db/IPFSClient.java
- Modification d'IPFS pour stocker les objets au sein d'un système de fichiers distribué, 250 lignes en Go, https://github.com/bconfais/go-ipfs/tree/common_backend_v1a
- Implémentation de notre protocole de localisation des objets au sein d'IPFS, 500 lignes en Go, https://github.com/bconfais/go-ipfs/tree/dns
- Implémentation d'un client IPFS pour le système d'exploitation RIOT, 900 lignes en C, https://github.com/bconfais/RIOT/tree/master/examples/ipfs_client, soumis pour être intégré à la version officielle du système d'exploitation : https://github.com/RIOT-OS/RIOT/pull/8889

Scripts de déploiement sur Grid'5000

- Script permettant de déployer Rados, Cassandra et IPFS, 1000 lignes en Python https://github.com/bconfais/benchmark/tree/master/Cloudcom2016
- Script permettant de déployer IPFS et RozoFS et permettant également d'évaluer notre couplage, 800 lignes en Python, https://github.com/bconfais/benchmark/tree/master/FEC2017
- Script pour déployer IPFS ainsi qu'un service DNS, afin d'évaluer notre protocole de localisation des objets, 900 lignes en Python, https://github.com/bconfais/benchmark/blob/master/dns
- Script pour établir un tunnel entre les plateformes Grid'5000 et FIT/IoT-Lab, déployer IPFS sur Grid'5000 et demander à un nœud sur la plateforme FIT d'écrire un objet, 600 lignes en Python, https://github.com/bconfais/benchmark/tree/master/iot-lab

Lors de cette thèse, nous avons utilisé 137 760 heures.cpu sur la plateforme Grid'5000. pour un total de 1377 heures d'expérimentation. Le dépôt Git de l'inria sur lequel les articles ont été écrits contient à l'heure actuelle plus de 3000 commits et a une taille d'environ 2 Go.

Table des figures

Liste des tableaux

Acronymes

AHHK *Alpert Hu Huang et Kahng*. 108, 111

ARM *Advanced RISC Machine*. 137

CDN *Content Delivery Network* ou Réseau de Diffusion de Contenus. 6, 33–35, 39, 41, 45, 46, 96

CFS *Cassandra FileSystem*. 33

CIFS *Common Internet FileSystem*. 30, 38

CRUSH *Controlled Replication Under Scalable Hashing*. 44, 54–56, 73, 90

DHT *Distributed Hash Table* ou Table de Hachage Distribuée. 10, 67, 69, 70, 72, 74, 76–79, 82, 85, 87, 92, 95, 103, 112, 113, 116, 117, 120–127, 129, 131, 132, 136

DNS *Domain Name System* ou Système de résolution des noms de domaine. 11, 100–103, 124, 125, 149

DNSSEC *Domain Name System Security Extensions*. 103, 149

FIT *Future Internet of Things*. 7, 11, 52, 115, 116, 137–141

HTTP *Hypertext Transfer Protocol*. 32

ICN *Information-Centric Networking* ou Réseau Centré sur l'Information. 36, 37

IoT *Internet of Things* ou Internet des Objets. 7, 11, 52, 115, 116, 137–141, 144, 145

IPFS *InterPlantery FileSystem*. 6, 7, 10, 39, 47, 51–54, 61–64, 66, 67, 69–80, 82–85, 87, 88, 98, 114–126, 128, 137, 138, 140, 141, 143, 144, 181

M2M *Machine-to-Machine*. 19

NAS *Network Attached Storage* ou Serveur de Stockage en Réseau. 7, 31, 47, 51, 52, 80, 82, 84, 85, 87, 88, 114–117, 119–124, 126, 141

NAT *Network Address Translation*. 139

NFS *Network FileSystem*. 15, 30, 38

OLSR *Optimized Link State Routing Protocol*. 89

OSD *Objet-based Storage Device* ou *Object Storage Daemon*. 54, 74

P2P *Peer-to-Peer* ou Pair à Pair. 21, 148

Rados *Reliable, Autonomic Distributed Object Store*. 6, 32, 44, 47, 51, 53, 54, 56, 58–67, 69–76, 79, 80, 90, 143

RENATER Réseau National de télécommunications pour la Technologie l'Enseignement et la Recherche. 93, 103, 111, 134, 138

SDN *Software-Defined Networking* ou Réseau Spécifié de manière Logicielle. 37

SSH *Secure Shell*. 139

TCP *Transport Control Protocol*. 66, 118, 120, 140, 141

TTL *Time To Live* ou Durée De Vie. 88, 102

WAN *Wide Area Network*. 31

YCSB *Yahoo Cloud System Benchmark*. 66, 67, 78, 118, 181

UNIVERSITE BRETAGNE LOIRE

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Bastien CONFAIS

Conception d'un système de partage de données adapté à un environnement de Fog Computing

A sharing data system adapted to a Fog Computing environment

Résumé

L'informatique utilitaire a évolué au fil des années pour aboutir à ce que nous appelons aujourd'hui le Cloud Computing. Pourtant, ces infrastructures ne sont pas adaptées pour répondre aux besoins de l'Internet des Objets ayant des besoins de calculs à faible latence malgré des ressources limitées. C'est pourquoi, en 2012, Cisco a proposé le paradigme de Fog Computing, consistant à répartir des serveurs sur de nombreux sites placés près des utilisateurs. Dans cette thèse, nous cherchons à créer une solution de stockage unifiée entre les différents sites de Fog. Notre première contribution consiste à évaluer si les solutions de stockage existantes peuvent être utilisées dans un tel environnement. Nous montrons que la solution de stockage InterPlanetary FileSystem (IPFS) reposant sur un protocole similaire à BitTorrent et une table de hachage distribuée (DHT) pour localiser les données est la plus prometteuse. Toutefois, le trafic réseau inter-sites généré impacte négativement les temps de lecture. Notre seconde contribution consiste à coupler IPFS au système de fichiers distribué RozoFS pour limiter ces échanges inter-sites dans le cas d'accès à des données stockées sur le site local. Enfin, notre dernier axe de recherche vise à localiser les données grâce à un protocole reposant sur un arbre des plus courts chemins, de façon à confiner le trafic réseau et à privilégier les nœuds atteignables avec une faible latence. Grâce à de nombreuses expérimentations sur la plateforme Grid'5000, nous montrons que le couplage à un système de fichiers réduit en moyenne de 34% les temps d'accès et que notre protocole de localisation permet un gain de 20% du temps de localisation des données.

Mots clés

Fog Computing, stockage distribué, localisation de données, arbre des plus courts chemins, Grid'5000, IPFS, RozoFS.

Abstract

Utility Computing has evolved for many years leading to the infrastructure we know today as Cloud Computing. Nevertheless, these infrastructures are unable to satisfy the needs of the Internet of Things which requires low latency computing despite limited resources. In 2012, Cisco proposed a paradigm called Fog Computing, consisting of deploying a huge number of small servers, spread on many sites located at the edge of the network, close to the end devices. In this thesis, we try to create a seamless storage solution between the different Fog sites.
Our first contribution consists in comparing existing storage solution and check if they can be used in a such environment. We show that InterPlanetary FileSystem (IPFS), an object store relying on a BitTorrent like protocol and a Distributed Hash Table is a promising solution. Nevertheless, the amount of network traffic exchanged between the sites to locate the data is important and has a non-negligible impact on the overall performance. Our second contribution consists in coupling IPFS with RozoFS, a distributed filesystem deployed on each site to limit the use of the DHT when accessed data are stored on the local site. Finally, we proposed to replace the distributed hash table by a location mechanism relying on a shortest path tree built on the physical topology, in order to contain the network traffic and to first request nodes at a close location, reachable with a low latency. By performing many experiments on the Grid'5000 testbed, we show that the coupling of IPFS with a Scale-Out NAS reduces by 34 % in average the access times and that our protocol to locate the objects reduces by 20 % the time to locate the data.

Key Words

Fog Computing, distributed storage, data localisation, shortest path tree, Grid'5000, IPFS, RozoFS.